BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DETECTION OF ENZYMES IN METAGENOMIC DATA

VYHLEDÁVÁNÍ ENZYMŮ V METAGENOMICKÝCH DATECH

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. STANISLAV SMATANA**
AUTOR PRÁCE

SUPERVISOR — **Ing. JIŘÍ HON**
VEDOUCÍ PRÁCE

BRNO 2017

Zadání diplomové práce/19850/2016/xsmata01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Smatana Stanislav, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Vyhledávání enzymů v metagenomických datech**
Detection of Enzymes in Metaganomic Data
Kategorie: Bioinformatika

Pokyny:

1. Seznamte se se základními pricipy metagenomiky a způsoby získávání metagenomických dat.
2. Seznamte se s existujícími přístupy a nástroji pro vyhledávání proteinových sekvencí.
3. Navrhněte nový nástroj pro vyhledávání enzymů v metagenomických datech. Do návrhu zakomponujte vhodný způsob filtrace podle specifikace funkčně důležitých aminokyselin.
4. Proveďte implementaci navrženého nástroje ve vhodném programovacím jazyce a ověřte jeho funkčnost na reálných datech.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího pokračování projektu.

Literatura:

- EDDY, S. R. Profile hidden Markov models. *Bioinformatics*. 1998, 14(9), 755-763. DOI: 10.1093/bioinformatics/14.9.755. ISSN 1367-4803.
- LEWIS, J. Fast template matching. *Vision Interface 95*. 1995, 120-123.
- Dále dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hon Jiří, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L. S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstract

This thesis presents specification and implementation of a system for detection of enzymes in metagenomic data. The detection is based on a provided enzyme sequence and its goal is to search the metagenomic sample for its novel variants. In order to guarantee that found enzymes truly have the desired catalytic function, the system employs a number of catalytic function verification methods. Their specification, implementation and evaluation is one of the main contributions of this thesis. Experiments have shown, that proposed methods reach sensitivity as high as 89%, specificity of 95%, values of AUC metric above 0.9 and average throughput of 1,203 verifications per second on regular personal computer. Evaluation of the system also led to discovery of a partial sequence of novel haloalkane dehalogenase enzyme in a metagenomic sample from soil. The implementation is able to work on a personal computer as well as on a grid computing environment.

Abstrakt

Hlavným cieľom tejto práce bolo navrhnúť a implementovať systém, ktorý by bol na základe vstupnej sekvencie enzýmu schopný vyhľadať v metagenomickej vzorke nové enzýmy s rovnakou funkciou. Aby bolo možné garantovať, že nájdené varianty skutočne katalyzujú rovnakú reakciu, je nutné ich katalytickú funkciu bližšie overiť. Jedným z hlavných prínosov tejto práce je práve návrh, implementácia a testovanie metód pre verifikáciu katalytickej funkcie. Experimenty ukázali, že navrhnuté metódy dosahujú senzitivitu 89%, špecificitu 95%, hodnoty metriky AUC nad 0,9 a v priemere dokážu na osobnom počítači vykonať 1 203 verifikácií za sekundu. Okrem toho bola počas testovania objavená čiastočná sekvencia nového enzýmu z rady halogénalkán dehalogenáz. Implementovaný systém je schopný fungovať na osobnom počítači, ako aj na distribuovanom systéme typu grid.

Keywords

metagenomics, enzymes, enzyme detection, catalytic function verification, catalytic site, novel enzymes, catalytic function, haloalkane dehalogenase, active site, catalytic residues

Klíčová slova

metagenomika, enzýmy, hľadanie enzýmov, overenie katalytickej funkcie, katalytické miesto, nové enzýmy, katalytická funkcia, haloalkan dehalogenáza, aktívne miesto, katalytické rezíduá

Reference

SMATANA, Stanislav. *Detection of Enzymes in Metagenomic Data.* Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Hon Jiří.

Detection of Enzymes in Metagenomic Data

Declaration

Here I declare, that I have created this master's thesis completely by myself under the supervision of Ing. Jiří Hon. I have listed all the literary sources and publications, which I have used to write this thesis.

.....
Stanislav Smatana
May 23, 2017

Acknowledgements

I would like to thank my supervisor, Ing. Jiří Hon, for his great help and support during the creation of my master's thesis. I would also like to thank experts from Loschmidt Laboratories for providing support and knowledge in the area of protein engineering.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

Contents

Chapter 1

Introduction

Enzymes represent a class of protein molecules, which is essential to all living beings. They are the "biological workhorses", which catalyze the vast net of chemical reactions vital for the survival of every cell [2]. Although, it is not only cells who benefit from their abilities.

In 2003, global sales of the enzyme-related industry reached 2.3 billion USD [34]. Industrial uses of enzymes range from the production of pharmaceuticals to the laundry detergent production, food applications and textile processing. All of these fields have a constant hunger for novel enzymes. Furthermore, because of the recent strong global political drive to move from fossil fuels into the area of renewable resources, more and more industries are expected to incorporate enzymes into their production processes [34]. Enzymes, unlike many traditional industrial methods, are independent from fossil fuels and are often able to produce biodegradable materials [17].

However, until recently, the search was limited to a small range of enzymes, which could be extracted from bacteria cultivated in laboratory conditions. It was estimated, that only 0.01% of bacteria could be grown in this way, and therefore the success in the search for new enzymes was becoming more and more rare [19]. The breakthrough came with the birth of metagenomic methods, which allow extraction of genetic material even from organisms, which could not be cultured. This, combined with modern high throughput sequencing technologies, allows to explore all the valuable information contained in biological sample using a computer [23].

The goal of my thesis is to exploit this novel source information and create a set of tools, that would be able to find new enzymes in a metagenomic sample. The search should be based on a provided sequence of some known enzyme, and its output should be a set of novel enzymes with the same catalytic function. While their function will be the same, found enzymes may have better chemical and physical properties and may be more suitable for practical use.

The proposed system consists of three main elements – read pre-processing, homology search and enzymatic function verification. While the first two are responsible for the searching process, the task of the enzymatic function verification is to guarantee that only enzymes, which truly have the desired catalytic function, will be outputted by the system. Design and implementation of these methods is one of the main contributions of this thesis, and is presented in the section 5.3. More information about the overall structure of the system is presented in its specification, located in the chapter 5.

All specified elements of the system were successfully implemented and their function was evaluated in a number of experiments. Details about the implementation are presented in the chapter 6 and the analysis of experimental results is provided in the chapter 7. Apart

from that, all theoretical information that is necessary in order to understand working principles of the system, is presented in chapters 2, 3 and 4.

Chapter 2

Genetic Information and Its Role in Living Cells

With the advent of light microscopy, it became quickly apparent, that living organisms are made out of small entities called *cells*. In a simplistic view, cell consists of a concentrated mixture of chemicals, called cytoplasm, which is encompassed by a plasma membrane [2].

While cells come in vastly different shapes, sizes and functions, there are two basic types of them – *prokaryotes* and *eukaryotes*. Prokaryotes are the less complicated of the two classes, and are commonly encountered in the form of bacteria. Eukaryotes, on the other hand, are much more complex and usually form multicellular organisms [2, 52].

In order to survive, a cell has to harvest chemicals provided by its environment and transform them into useful molecules. This is possible thanks to a vast and tightly controlled network of chemical reactions directed by a special class of protein molecules called enzymes [2].

It is apparent that cell needs some way of storing information on how to produce enzymes and how to regulate its reaction networks. Moreover, cells have ability to copy themselves, and therefore this storage has to contain information about the structure of the whole cell. All of these instructions are stored in the form of a special molecule called *deoxyribonucleic acid*, or in short, the DNA. While in prokaryotes, the deoxyribonucleic acid floats freely in the cytoplasm, in eukaryotes, it is encompassed within a special structure called *nucleus* [52].

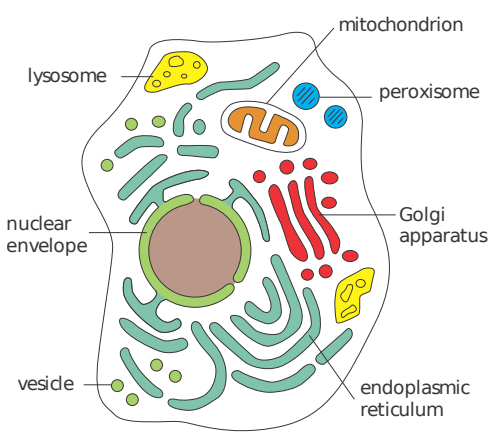

Figure 2.1: The eukaryotic cell contains a number of structures that are absent in its prokaryotic counterpart. These are called organelles and are shown on this diagram. The most important organelle for the purpose of this thesis is the nucleus, which is surrounded by the nuclear envelope and contains the DNA. In prokaryotes, the DNA floats directly in cytoplasm. Image was taken from a book by Alberts et al. [2].

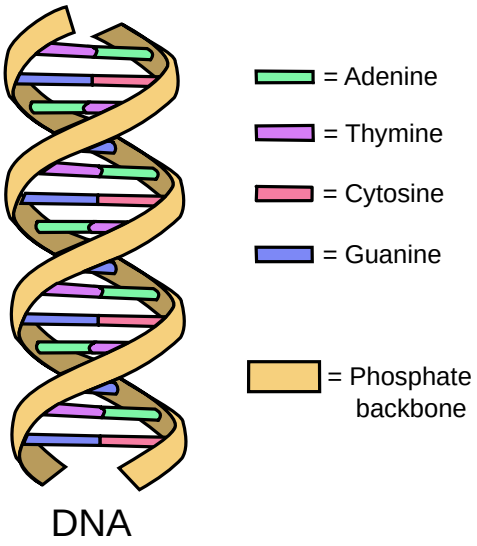

Figure 2.2: The DNA in cells has two complementary strands which form a double helix. This is possible thanks to the Watson-Crick base pairing principle, which states that adenine can only pair with thymine and guanine with cytosine. The image was taken from Wikimedia Commons and is distributed under public domain with no restrictions.

2.1 Structure and Content of Deoxyribonucleic Acid

The deoxyribonucleic acid (figure 2.2) is a biological macromolecule created by chaining of simpler molecules called *nucleotides* [52]. Chemically, each nucleotide contains phosphate, sugar and a nitrogenous base. While sugar and phosphate create so called *sugar-phosphate backbone*, which keeps the whole DNA together, the nitrous base is the part which distinguishes nucleotides of one type from another. In efect, the nitrous base serves as the carrier of information [2]. Bases come in four basic types – *adenine* (A), *thymine* (T), *cytosine* (C) and *guanine* (G). From the information perspective, DNA could be seen as a long string on the alphabet $\{A, C, T, G\}$. Moreover, it is important to add, that ends of the "DNA string" are distinguishable by their chemical properties and are commonly labeled as the 3' end and 5' end [2, 52]. Thanks to this, it is easy to denote direction of a movement relative to the molecule.

However, it is unusual for the DNA to reside in form of a single string. In living organisms, it is much more common to find it as a double helix composed of two DNA strings. This is possible thanks to the ability of nucleotides to form hydrogen bonds with other nucleotides. Although, this bonding has strict rules – A can be only bonded with T and G with C. This fact was first discovered by biologists Watson and Crick, and therefore it is named after its discoverers the *Watson-Crick base pairing* [52]. Thanks to it, the two chains of the DNA are complementary to each other.

Apart from the obvious benefit of redundancy, the complementarity allows each string to be used as a template for synthesis of its counterpart. This synthesis process is called *replication* and is performed by a large protein complex known as *replisome* [2]. The DNA replication is crucial for the cell division and serves as the primary means of information transfer from parent cell to its children.

One of the most important information contained within the DNA are instructions for protein creation. Protein, like the DNA, is a linear sequence of elements, however, unlike the DNA, these elements are not nucleotides, but molecules called *amino acids*. While there are only 4 types of nucleotides, amino acids come in 20 standard variations. Therefore, the cell has to somehow translate string in 4 symbol alphabet of nucleotides into a string in

the 20 symbol alphabet of amino acids. This is done by the transcription and translation processes, which will be described in the next chapter. Input of these processes is a sequence of nucleotides encoding one or a group of related proteins called *gene* [52].

While in prokaryotes, the gene is informationally dense and contains only nucleotides relevant for the encoding of a protein, the eukaryotic gene is interspersed with long non-coding sequences called *introns*. In fact, there are typically more non-coding *introns*, than there are their coding counterparts called *exons* [3]. These non-coding parts are cut away and degraded later in the process.

However, not all DNA is used for protein coding. There are also regions used for regulation of gene expression, regions encoding functional RNA and regions of repetitive sequences, which are introduced by mobile genetic elements called *transposons* [3].

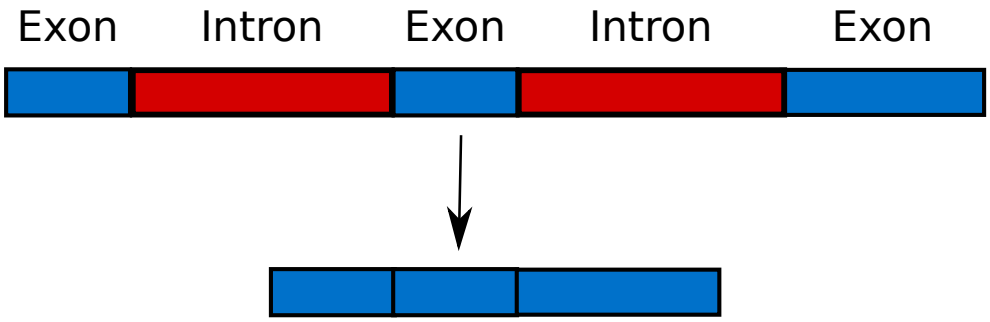

Figure 2.3: The eukaryotic gene consists of coding exons (blue color) interspersed with non-coding introns (red color). Later, after the gene has been transcribed, the introns are removed and exons are glued together in a process called *splicing*. Result is a continuous sequence of coding nucleotides.

2.1.1 Representation of Deoxyribonucleic Acid in a Computer

A sequence of nucleotides can be transferred from a DNA into a computer by process called *sequencing*. However, contemporary sequencing methods are not able to transfer whole molecule in one chunk and require it to be split into smaller parts called *fragments*.

While fragments can be processed by the sequencing instrument, it is not possible to transfer their full sequence, but only small part of it. These parts are called *reads*, and their typical length in contemporary methods ranges from 25 nucleotides to approximately 1000. While older technologies provided longer read lengths, newer methods, called *Next Generation Sequencing Methods*, provide shorter reads with higher precision and lower price [38].

Currently, there are two strategies of extracting reads from fragments – *single read sequencing* and *paired read sequencing*. In the single read mode, each fragment is sequenced only from one side, resulting in one read per fragment. In contrast, the paired read sequencing (figure 2.4) extracts reads from both sides of the fragment, resulting in two reads per fragment. The information about distance between reads in pair helps to improve results of genome assembly [39]. The process of genome assembly is described in section 4.1.

Apart from the sequence itself, the output of an automated sequencer contains per base estimates of sequencing quality (i.e. belief in correctness of a given base). The most common data format used to store both quality and content of reads is the *fastq file format* (figure 2.5) [13]. In fastq format, nucleotide sequences are denoted as strings over the

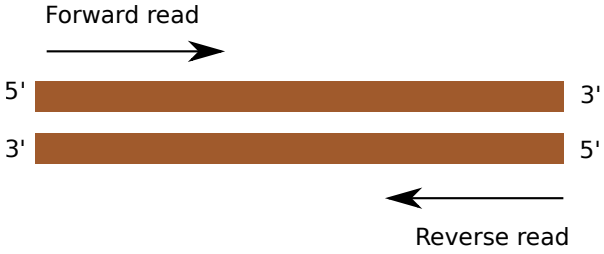

Figure 2.4: In paired read sequencing, a fragment is sequenced from both sides. The distance between the pair of resulting reads is known, and this fact helps with genome assembly.

alphabet {A, C, T, G}, and their quality values are encoded as character strings of the same length. Quality values are typically used for data pre-processing and afterwards, when the information about quality is no longer needed, a fastq file can be transformed into corresponding *fasta* file (figure 2.6). The *fasta* format is very similar to *fastq*, but omits the quality information, and therefore requires less storage space.

```
@SRR014849.1 EIXKN4201CFU84 length=93
GGGGGGGGGGGGGGGGCTTTTTTTGTTTGGAACCGAAAGG
GTTTTGAATTTCAAACCCTTTTCGGTTTCCAACCTTCCAA
AGCAATGCCAATA
+SRR014849.1
EIXKN4201CFU84 length=93 3+&$#""""""""""""7F@71,'";C?,B;?6B;:EA1EA
1EA5'9B:?:#9EA0D@2EA5':>5?:%A;A8A;?9B;D@ /=<?7=9<2A8==
```

Figure 2.5: A record of one read in fastq format. First line of each record starts with the character '@' and continues with the record title. Lines after it contain the sequence written in ascii characters. After the sequence, the line with title can be repeated, but has to start with the character '+' instead of '@'. Finally, last lines of the record contain the base quality estimates. Qualities are encoded as characters and the length of quality string has to match the length of sequence [13]. Figure was taken from an article by Cock et al. [13].

```
>SRR014849.1 EIXKN4201CFU84 length=93
GGGGGGGGGGGGGGGGCTTTTTTTGTTTGGAACCGAAAGG
GTTTTGAATTTCAAACCCTTTTCGGTTTCCAACCTTCCAA
AGCAATGCCAATA
```

Figure 2.6: A record of one read in fasta format. First line starts with the character '>' and contains the name of the sequence. Following lines contain the sequence data. The figure was created based on a figure from article by Cock et al. [13].

2.2 Protein Biosynthesis

Proteins are one of the most vital elements of cells, and every cell has to be able to produce them as needed. Instructions for their building are encoded in the DNA. The process of protein synthesis has two main steps – *transcription* (explained in the section 2.2.1) and

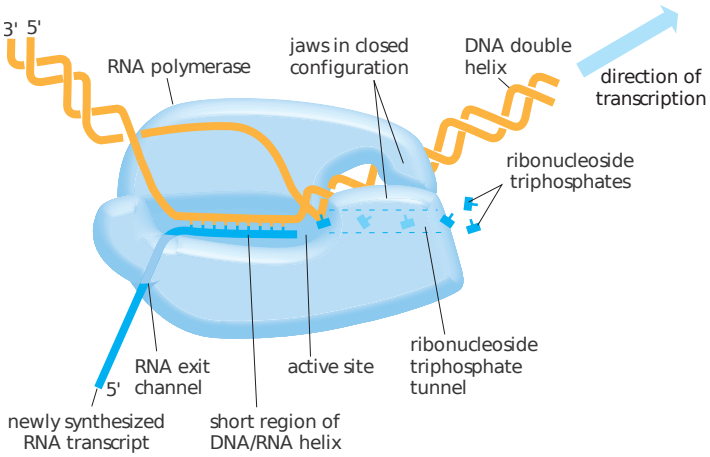

Figure 2.7: RNA-polymerase performing the transcription process. The newly synthesized RNA string floats freely, while the unwinded DNA behind the RNA-polymerase rapidly reestablishes its prior double-helical form. Note that the transcription is always performed in the 3'-5' direction. Figure was taken from book by Alberts et al. [2].

translation (explained in the section 2.2.2). However, these processes are not the only way in which genetic information can flow. All the possible genetic information transfers are described by the *central dogma of molecular biology*, which is the subject of section 2.2.3.

2.2.1 The Process of Transcription

In the first step of protein synthesis, a gene is *transcribed* from DNA to a very similar type of molecule called *ribonucleic acid* (RNA). Since the RNA molecules produced in cell can have different functions, the type of RNA used in transcription is distinguished by the name *mediator* RNA (mRNA).

As in the case of DNA, the cornerstone of every RNA molecule is a sugar-phosphate backbone. However, the RNA molecule is synthesized using a different type of sugar called *ribose* and usually comes in a form of a single string, which is significantly shorter than typical DNA molecule [2]. Morover, the RNA does not contain the base thymine, but a different kind of base called *uracil* instead [2]. Like thymine, uracil pairs with adenine, and therefore enables the synthesis of complementary RNA for a given DNA. In effect, the RNA could be viewed as a string over the alphabet $\{A, C, U, G\}$.

The transcription of DNA into RNA is performed by an enzyme called *RNA-polymerase*. In prokaryotes, the RNA-polymerase binds to the start of transcribed segment and unwinds a small portion of the DNA. Afterwards, it moves along one strand, continues to unwind the DNA in front of it and synthesizes the complementary RNA chain. Synthesized RNA is not bound to the DNA and floats freely in the surrounding space. The DNA behind moving polymerase rapidly reestablishes its prior double-helical form. This setup allows multiple polymerases to proceed immediately after each other (i.e. form a "convoy") and transcribe one region at the same time [2].

In order to transcribe genes correctly, the RNA polymerase has to somehow determine where the gene starts and where it ends. This is denoted by two special nucleotide sequences on the DNA called *promoter* and *terminator* [52]. During its lifespan, the RNA-polymerase randomly collides with the DNA. Upon collision, it forms a bond with it, however, unless this bond is formed with the promoter sequence, it is not strong enough and rapidly disintegrates. The promoter region is asymmetrical and ensures that the transcription is conducted only in one direction and only on one DNA strand. The strand and direction can be different for each gene. On the opposite end, the terminator results in polymerase being detached from the DNA and the synthesized RNA being released into the cytoplasm [3].

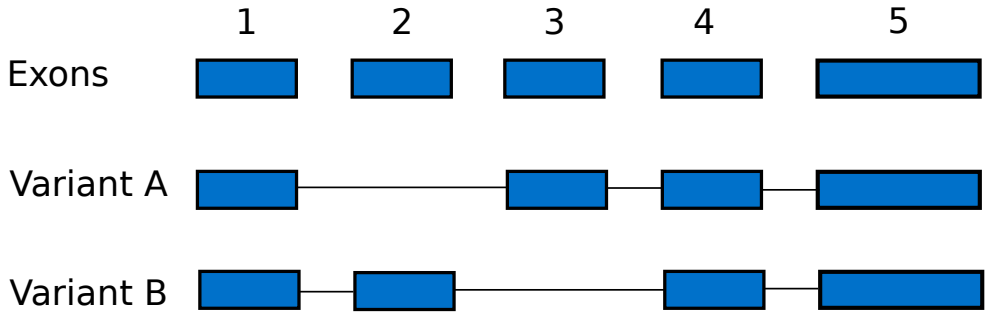

Figure 2.8: Alternative splicing of an eukaryotic gene. The set of exons can be assembled in multiple ways. For instance, the variant A has the second exon absent while the third exon is present. In contrast, the variant B has it the other way around. Thanks to the alternative splicing, one eukaryotic gene can code for a number of similar proteins.

While in the case of prokaryotes, the released mRNA directly proceeds to the *translation* step, the mRNA in eukaryotes has to be transported out of the nucleus through nuclear pore. Apart from that, it also undergoes multiple post-processing steps. Many of them increase the stability of the molecule and allow it to be safely transported. However, there is a significant post-processing step called *splicing* [2]. As it was mentioned earlier, the typical eukaryotic gene has its coding regions interspersed with non-coding regions called introns. These are removed in the splicing step by a protein-RNA complex called *spliceosome* [3]. The resulting set of exons can be assembled in a number ways. This phenomenon is called *alternative splicing* and it allows a single eukaryotic gene to encode multiple similar proteins [2].

It may be argued, that the transcription stage of protein synthesis is unnecessary, because it would be simpler for the cell to synthesize proteins directly from the DNA. However, the transcription allows to create multiple copies of the synthesis template, which in turn allows massive parallelization of protein production. Also, the mRNA serves as a basic regulator of protein synthesis, because different genes tend to be transcribed into different amounts of mRNA [2]. In the following section, I will describe the second step of protein synthesis called *translation*.

2.2.2 The Process of Translation

In order to create a protein, its gene, written in the alphabet of nucleotides, must be rewritten into the alphabet of protein building blocks – the amino acids. Every amino acid is encoded by a triplet of nucleotides called *codon* [3]. This leads to 4^3 or 64 possible coding words. As a result, the mapping between 64 codons and 20 amino acids is not bijective and some amino acids are encoded by more than one distinct codon. This mapping is commonly referred to as the *genetic code* [3].

In reality, it is provided by a special adapter molecule called *transfer RNA* or the tRNA [2]. It is an L-shaped molecule that contains an "anticodon" – a sequence complementary to a given codon [3]. Thanks to the ability of nucleotides to form bonds with other nucleotides, the anticodon can form a bond with its complementary codon. Moreover, because the opposite end of the tRNA can bind amino acid corresponding to its anticodon, the tRNA molecule provides a full physical realization of the genetic code mapping. This amino acid, however, is not a part of the molecule and has to be added via a chemical

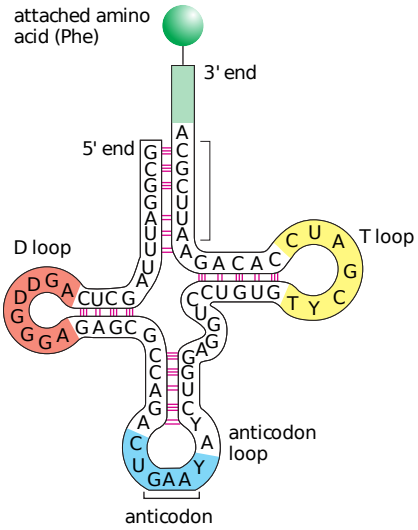

Figure 2.9: Illustration of the tRNA molecule. The most important parts of the tRNA are the 3' end and the anticodon. While the 3' end binds an amino acid, the anticodon is bound to the RNA. The binding is realized according to the Watson-Crick base pairing. For example, the anticodon "GAA" on the figure will be bound to the codon "CTT". Figure was taken from book by Alberts et al. [2].

reaction mediated by one of the *aminoacyl-tRNA-synthetase* enzymes [2]. Usually, there is a different enzyme for every amino acid [3].

While the tRNA provides the necessary mapping, it is not able to form the polypeptide chain on its own. This reaction is conducted by a complex molecule consisting of RNA and proteins called *ribosome*.

Since there are 3 nucleotides encoding one amino acid, there are 3 possible offsets, from which the translation can begin. These offsets are called *reading frames*. If we take into account, that genes can be stored on both strands of the DNA[1], there is a total number of 6 possible reading frames. While in theory, every reading frame could encode a different protein, living organisms usually use only one reading frame per gene [3].

As in the case of transcription, the beginning and end of translation is signaled by one of the distinct nucleotide sequences – the start codon (AUG) and one of the stop codons (UAA, UAG, UGA)[2] [2]. While the start codon results in a special *initiator tRNA* with the amino acid methionine being added to the beginning of the chain, the stop codons result only in polypeptide chain release and ribosome disintegration without further addition of amino acids into the chain.

Like in the case of RNA-polymerases, multiple ribosomes can translate one mRNA molecule at the same time. In case of prokaryotes, where the mRNA does not have to be transported from the nucleus, the translation can begin even before the transcription is finished. This results in a significant speedup of prokaryotic protein synthesis [3].

2.2.3 The Central Dogma of Molecular Biology

In the preceding section, it was shown, that the typical flow of genetic information in living organisms begins with a nucleic acid (e.g. DNA) and ends with some protein or some functional RNA. Apart from that, there are viruses that can rewrite their RNA into host's DNA, and, in laboratory conditions, it is even possible to synthesize proteins directly from the DNA [37, 51]. However, genetic information never flows from proteins back to

[1] Genes from different strands are stored in opposite direction because the transcription can only proceed from the 5' end towards the 3' end of the molecule.

[2] It is important to keep in mind that the promoter, terminator, start codon and stop codon are all different sequences.

any nucleic acid. This important statement is known as *the central dogma of molecular biology* [51].

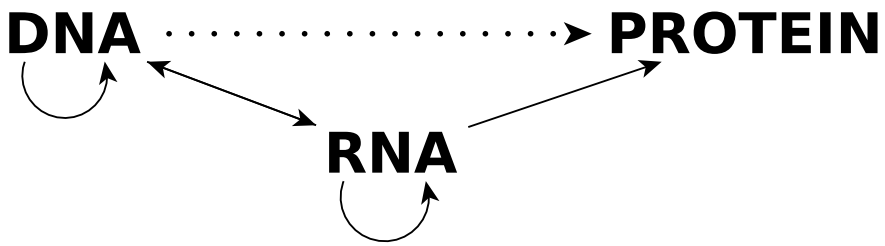

Figure 2.10: The central dogma of molecular biology. The typical flow of genetic information goes from a nucleic acid to a protein sequence. Transfer from DNA to DNA (replication), RNA to RNA (RNA viruses) and RNA to DNA (retroviruses) is also possible. However, transfer from protein to RNA or DNA was never observed. Transfer from DNA to a protein is denoted by a dotted line, because it was only observed in laboratory conditions outside of a living cell [37].

Chapter 3

Proteins and Their Biological Significance

Proteins are one of the most important elements of living organisms. They form the largest part of the dry mass of all cells and perform a great number of functions necessary for cell's survival [3]. These functions are ranging from a transport of molecules and their storage to chemical reaction catalysis and gene regulation [2].

While the section 3.1 focuses on the relationship between the function of an enzyme and its structure, the second section of this chapter (3.2) looks more deeply into the catalytic function of proteins and describes a very important class of proteins called *enzymes*.

3.1 Structure of Proteins

Similarly, like in the case of the DNA, proteins are strings composed of elementary building blocks. However, the building blocks of proteins are *amino acids* and not nucleotides. These elements are bonded together by a chemical bond called *peptide bond* and therefore, the string of amino acids is sometimes referred to as the *polypeptide* [3].

The peptide bond is flexible[1] and allows arbitrary rotation of its elements. This allows the string to be freely folded. However, proteins in living organisms do not randomly change their fold from one to another. On the contrary, they have a stable shape. Stability of a given shape is ensured by additional chemical bonds between side chains of amino acids, which are weaker and more rigid than the peptide bond[2] [3]. Polypeptide always folds into form which has the lowest energy called *conformation* [2]. However, if the stabilizing bonds are destroyed, for example by heat or some acid, the protein unfolds back into the form of a string. This process is called *denaturation* [3].

A protein can have a set of stable conformations and it can change its state from one to another. Sometimes, this change is induced by the binding of a foreign molecule onto a special region on the protein surface called *binding site*. This binding is realized by the same type of bonds as bonds, which stabilize the protein conformation. Since these bonds are weak, there has to be a great number of them in order to properly stabilize the molecule on a binding site [3]. As a result, the surface of the foreign molecule (referred to as the

[1] Chemically speaking, the peptide bond is a type of a covalent bond [2].

[2] These bonds are non-covalent and are one of the three types – hydrogen, electrostatic attraction or Van Der Waals [2].

Figure 3.1: An example of a polypeptide chain folded into conformation. The polypeptide consists of a single string of amino acids, which automatically folds into the shape with lowest energy. The exact shape depends on the chemical and physical properties of polypeptide's amino acids. This example illustrates conformation of catalytic domain of botulinum neurotoxin serotype A (PDB accession 4ZJX). The side chains of amino acids are not shown. The figure was generated by software PyMol [15].

ligand) has to closely follow the surface of the binding site. This ensures that binding of ligands is highly specific [2].

3.1.1 Representation of Peptide Sequences in Computers

Since, like the deoxyribonucleic acid, protein is a sequence of successive elements, the same file format can be used for its storage. However, while there are only 4 types of nucleotides, the group of naturally occurring amino acids has 20 members. Because of this, the International Union of Pure and Applied Chemistry developed a standard one-letter notation of different amino acids, which I will refer to as the IUPAC alphabet [1]. Figure 3.2 shows an example of a protein stored in the fasta format using the IUPAC alphabet.

```
>5LKA:A|PDBID|CHAIN|SEQUENCE
SLGAKPFGEKKFIEIKGRRMAYIDEGTGDPILFQHGNPTSSYLWRNIMPHCAGLGRLIACDLIG
MGDSDKLDPSGPERYAYAEHRDYLDALWEALDLGDRVVLVVHDWGSALGFDWARRHRERVQGIA
YMEAIAMPIEAADLPEQDRDLFQAFRSQAGEELVLQDNVFVEQVLPGWILRPLSEAEMAAYREP
FLAAGEARRPTLSWPRQLPIAGTPADVVAIARDYAGWLSESPIPKLFINAEPGALTTGRMRDFC
RTWPNQTEITVAGAHFIQEDSPDEIGAAIAAFVRRLRPAHHHHHH
```

Figure 3.2: Haloalkane Dehalogenase linB (PDB accession 5LKA) amino acid sequence in fasta format. Amino acids are encoded using the IUPAC alphabet.

.

3.2 Enzymes and Metabolic Pathways

Binding sites are the key elements of an important class of proteins called *enzymes*. Enzymes have special binding sites called *active sites*. Ligands bound to them are called

substrates and are transformed by the catalytic function of the enzyme into different chemical compounds called *products*. This process is conducted by some of the amino acids, which are located at the active site and are commonly called *catalytic residues* [3]. The exact type of amino acids involved is depended on the type of catalyzed chemical reaction. In the absence of enzymes, these reactions would be slow or totally impossible. Chemically speaking, enzymes lower the energy needed for a given chemical reaction to start (i.e. the *activation energy*) [2].

However, complex chemical reactions often require cooperation of multiple enzymes. In this setting, an output of one enzyme becomes the input of some other one, and in turn, the output of the second enzyme becomes an input of the third, etc. These chains of enzymes are called *metabolic pathways* and may include an arbitrary number of enzymes and have an arbitrary level of branching [2, 52].

Enzymes are not only important in living organisms, but also in industrial applications. One such example is a class of enzymes called *haloalkane dehalogenases*, which I use in my master thesis as the model enzyme class.

Halogenated compounds are compounds containing one of the halogen atoms - fluorine, chlorine, bromine, iodine and astatine. These are often by-products of industrial chemical reactions and can be toxic and carcinogenic to living organisms [55]. Haloalkane dehalogenases catalyze dehalogenation, or in other words, the removal of halogen atoms. Resulting compounds are usually less toxic then their halogenated precursors [55].

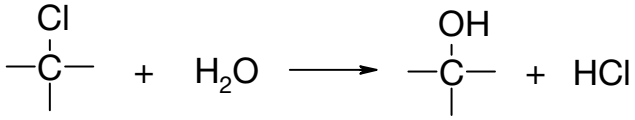

Figure 3.3: Reaction scheme of hydrolytic dehalogenation. Chlorine is detached and substituted by an OH group from a water molecule. The result is an alcohol and the hydrochloric acid. This reaction would not be possible without the presence of a dehalogenase. Figure was taken from an article by Van Pee [55].

Chapter 4

Methods of Genetic Information Processing and Analysis

After a genetic material was successfully transferred into a computer, it can undergo a further analysis. Algorithms for this analysis belong to the domain of bioinformatics. In this section, I will describe methods, which solve two important problems – *genome assembly* and *homology search*.

As the sequencing process is unable to read a whole DNA molecule and its output is usually very fragmented, bioinformaticians have developed strategies to assemble fragments back into the original sequence. These are discussed in the section 4.1.

Apart from the original sequence, biologist may be interested in identification of proteins in newly obtained reads. This is the goal of homology search methods and alignment algorithms described in section 4.2.

4.1 Assembly of Genomes

Genomic assembly is a process of reconstruction of an original genome from a set of short fragments called reads. In practice, there are two main classes of algorithms, which deal with this problem - the Overlap-layout-consensus class of algorithms (section 4.1.2) and the class of algorithms based on a De Bruijn graph (section 4.1.1) [33].

4.1.1 Methods Based on De Bruijn Graph

Contrary to the intuition, the first step of an assembly using a De Bruijn graph is further fragmentation of reads [33]. Firstly, reads are fragmented into k-mers (strings of the equal length k). This is performed using a sliding window approach, in which a window of length k moves base by base from one end of a read to the other and extracts a string of length k at every position. In effect, k-mers in the resulting set share a lot of overlap. Next, these k-mers are further subdivided into (k-1)-mers (strings of the length k-1). This results in exactly two (k-1)-mers per every k-mer: a left (k-1)-mer and a right (k-1)-mer.

The set of all (k-1)-mers is used for the construction of a De Bruijn graph, which is the central data structure of these methods. A De Bruijn graph is a directed multigraph[1], where each node represents a (k-1)-mer. It is important to note, that there is exactly one node for every distinct (k-1)-mer even when there are multiple occurrences of it. For each original

[1]Multigraph is a type of graph where multiple instances of each edge are allowed.

k-mer, an edge from the node representing its left (k-1)-mer to the node representing its right (k-1)-mer is added. Please refer to the figure 4.1 for an example of a De Bruijn graph.

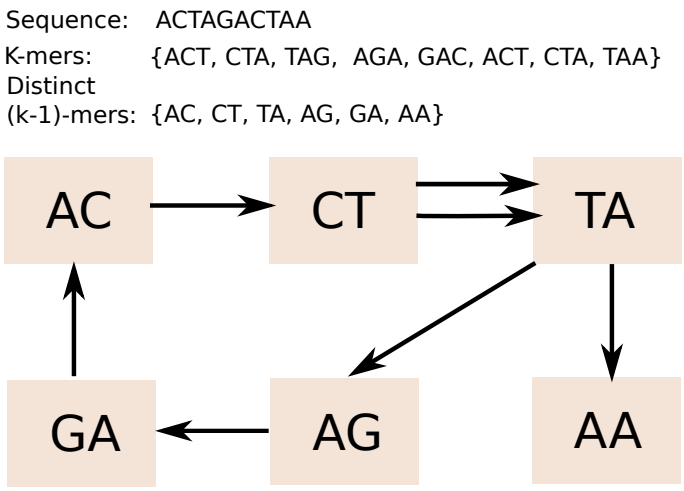

Figure 4.1: Example De Bruijn graph for the sequence ACTAGACTAA. In this example, the sequence ACTAGACTAA is split into reads which exactly correspond to the set of k-mers for k=3. Each of these is divided into left and right (k-1)-mer. Directed edge is added in between every left and right (k-1)-mer node. The original sequence corresponds to an Eulerian walk (a walk which visits every edge exactly once) through the De Bruijn graph. In this example, it is the walk through nodes AC, CT, TA, AG, GA, AC, CT, TA and AA. However, the situation is more complex in a real assembly process, because k-mers do not correspond to reads exactly, and the graph is not guaranteed to be Eulerian [47].

The assembly problem in a De Bruijn graph is reduced into the problem of finding an Eulerian walk through the graph [47]. An Eulerian walk is a type of walk in which every edge is visited exactly once. There exists a number of algorithms, such as Fleury's algorithm and Hierholzer's algorithm, which can find an Eulerian path in a given graph.

However, in real situations, a De Bruijn graph is not guaranteed to be Eulerian. For this reason, modern assemblers usually output partial sequences, called *contigs*, which were unambiguously assembled, instead of a whole genome assembly. Regions containing ambiguities need further analysis and the situation may even need a help of a human expert.

4.1.2 Overlap-Layout-Consensus Class of Assembly Methods

Apart from methods based on a De Bruijn graph, there is another major family of assembly algorithms called Overlap-layout-consensus methods (OLC methods). These methods were very popular in the past, but with the advent of Next Generation Sequencing, the rate of their usage has dropped significantly [39, 33]. The main reason behind their decline is their computational cost, which is caused by the need to find overlaps between all reads in a dataset [33].

Overlaps have to be found in order to construct an overlap graph, which is the central data structure of OLC methods. Similarly, like a De Bruijn graph, the overlap graph has directed edges. However, unlike the De Bruijn graph, the nodes of this graph represent whole reads and not only substrings of reads [39]. The edges of the overlap graph connect nodes which share a common overlap longer then a set threshold. The direction of an edge goes from a read with an overlapping suffix to a read with an overlapping prefix. Both,

the overlap calculation and the overlap graph creation, are a part of the *overlap* step of an OLC method.

After the graph has been constructed, its content is used in the *layout* step to assemble a genome. In the context of an overlap graph, the solution is a walk through the graph in which every node is visited exactly once [33]. This type of walk is called the Hamiltonian walk. Unfortunately, the problem of finding a Hamiltonian walk is NP-hard [33].

Finally, the overlapping parts of the sequence are aligned, and a final consensus sequence is determined for every one of them. This is the purpose of the consensus step, which is the final step of the whole method.

4.2 Sequence Alignment and Homology Search

Most of the modern biological applications, like the analysis of transcriptome, the study of methylation patterns, or the search for related sequences, require methods of sequence alignment and homology search [31].

The cornerstone of these methods is a dynamic programming alignment algorithm proposed by Needleman and Wunch and its later variant for local alignment proposed by Smith and Waterman. Both of these are described in the section 4.2.1.

While the dynamic programming method is able to provide the optimal alignment of two sequences, it is not fast enough to be used for a large-scale comparison of sequences in a database. The solution to this problem came in a form of an heuristic algorithm called Basic Local Alignment Tool, described in section 4.2.2, and later, in a form of methods based on the Profile Hidden Markov Model, which are the subject of section 4.2.3.

4.2.1 Dynamic Programming Algorithm for Global and Local Sequence Alignment

In general, a dynamic programming algorithm solves problems by their subdivision into smaller subproblems. These subproblems are easily solved and their partial results are then evaluated to solve the original problem. This approach was firstly used for exact sequence alignment by Needleman and Wunch [41].

Their algorithm represents the whole problem as a matrix of integers (figure 4.2). Given an arbitrary sequence A and sequence B, the matrix contains as many columns as there are symbols in the sequence A; and as many rows as there are symbols in the sequence B. However, in practice, one extra column and one extra row is added. Alignment is represented as a path in this matrix. Diagonal movement corresponds to comparison of symbols, movement from left to right corresponds to the insertions of gaps into the sequence B and vertical movement corresponds to insertion of gaps into the sequence A [56].

The first row and the first column of the matrix are filled by multiples of a gap penalty value δ. Rest of the matrix is filled row by row using the values that were calculated in previous steps. Formally, a value of matrix cell in the row i and the column j is calculated as

$$M_{i,j} = max(M_{i,j-1} - \delta, M_{i-1,j} - \delta, M_{i-1,j-1} + S(A_i, B_j))$$

where δ is the gap penalty and S is a scoring function which provides a comparison score of two symbols [26]. Gap penalties were introduced, because insertions and deletions of amino acids are rare, while substitutions of one amino acid for another are much more common [56].

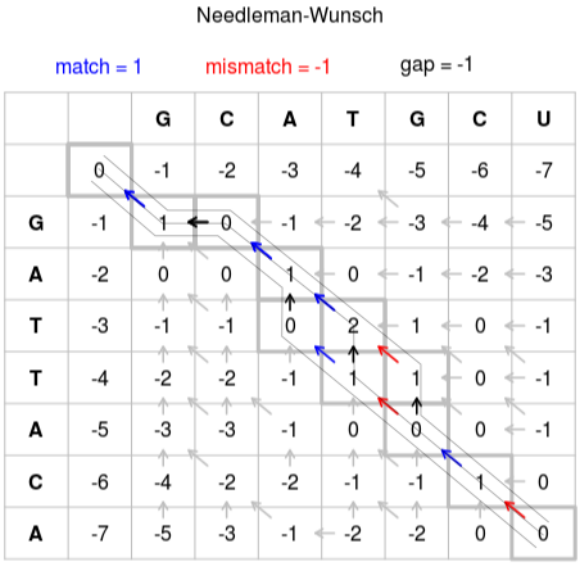

Figure 4.2: Dynamic programming matrix of the Needleman-Wunch global alignment algorithm. In the first step, the first row and column is initialized to multiples of the gap penalty (-1 in this case). Matrix is then filled row by row until the bottom right cell is reached. In the final step, the algorithm backtracks through the matrix, obtaining the resulting alignment. This figure was taken from Wikimedia Commons and is distributed under public domain with no restrictions.

The preceding example have illustrated the use of the *constant* gap penalty model. However, this model does not represent reality, because the start of a new gap is less probable then extension of an existing one [26]. This fact led to the introduction of the *affine gap penalty model*, which uses different penalties for the gap creation and the gap extension. Given a gap creation penalty γ and a gap extension penalty δ, the total penalty of a given gap can be calculated according to the formula

$$W = \gamma + \delta(k - 1)$$

where k is the length of this gap [56].

The last element of the alignment algorithm is a scoring function S. In its simplest form, it may simply evaluate to 1 if the two symbols are equal and to 0 if they are not. However, in practice, a special substitution matrix may be used in its place. This is done in order to achieve higher quality alignment [56]. The two most commonly used families of scoring matrices for an alignment of amino acid sequences are the Point Accepted Mutation (PAM) family of matrices and the Blocks Substitution Matrix (BLOSUM) family of matrices. Both of these families were empirically derived [26].

In the final step, the algorithm backtracks through the dynamic algorithm matrix from the bottom rightmost position towards the top leftmost position using the highest scoring path. This path represents the optimal alignment that was found by the algorithm [26]. Typically, both the alignment and an alignment score, which is the value of the rightmost cell on the bottom, are outputted.

The described algorithm is commonly referred to as the *global sequence alignment*, because it tries to align both sequences over their whole length. However, it is unsuitable in situations, where the user wants to find occurrences of a shorter sequence in a longer sequence (e.g. when the task is to find a gene in a whole genome) [26]. For this purpose, the *semi-global alignment* algorithm was introduced. The semi-global alignment follows the same steps as the global alignment, but the first row and column are initialized to 0, instead of the multiple of a gap penalty, and insertions of gaps at ends of sequences are not penalized. In effect, the algorithm allows addition of leading and trailing gaps without

penalization, and therefore allows arbitrary positioning of the shorter sequence along the longer one [26].

The last modification of the global alignment algorithm, called the *local alignment* algorithm, was proposed by Smith and Waterman [53]. As in the case of the semi-global alignment, the first row and column of the dynamic programming matrix is initialized to 0. However, unlike the global and semi-global alignment, the local alignment does not allow negative scores in matrix cells and saturates negative values to zero. Another difference is in the backtracking method. While the global and semi-global alignment always backtracks from the bottom rightmost cell of the matrix, the local alignment backtracks from all cells with the maximal value. Moreover, the backtracking stops, when the first zero value is reached [53].

4.2.2 The Basic Local Alignment Search Tool Algorithm

While the local alignment algorithm can be used for search in a big database of sequences with optimal results, its time complexity prevents its effective use for this task. This was one of the main motivations, which led to the invention of the Basic Local Alignment Search Tool (BLAST) algorithm [4].

BLAST is a heuristic algorithm, which performs search over a database of sequences using an input sequence in order to obtain its homologues. Contrary to the case of dynamic programming, the result is not guaranteed to be optimal, however, it has reasonable quality and the searching process is significantly faster [4]. The algorithm has a number of steps which will be described in the rest of this section.

In the first step, the input sequence is divided into a number of smaller sequences of constant length W called words. The division is performed using a sliding window approach, in which a window moves base by base from one end of the input sequence to the other [26]. At its every position, a sequence of length W is extracted. The parameter W is one of the input parameters of the whole algorithm. Since it is more probable to find occurrences of a small string, the smaller the value of parameter W is, the more matches will be found by the algorithm [4].

Next, for every word obtained in the previous step, a set of alternative words is created by various substitutions. These alternatives are compared with the original word using a chosen scoring matrix. Only alternatives, with the comparison score above a threshold T, are kept in the set, while the others are discarded [26]. T is another parameter of the algorithm, which regulates how similar a found sequence has to be in order to be reported [4].

Afterwards, all the words from the set of both original and alternative words are searched for in the database. Every match serves as a basis for an alignment between the input sequence and a sequence in the database. This alignment is iteratively extended on both of its sides. After every extension, an alignment score is calculated using the scoring matrix, and if it is higher then a predefined threshold S, the alignment is added into the set of *high scoring pairs*. The extension continues until the score drops below another threshold X [26].

In the final step, the high scoring pairs are filtered with respect to their statistical significance. The statistic used in this process is called E-value, and it represents the expected number of sequences with the score S or higher, which would be found in the database by a mere chance [26, 4]. The score S represents the alignment score of a high scoring pair. The user can choose the maximal allowed E-value for a search result. High scoring pairs which pass this test are the resulting output of the BLAST algorithm.

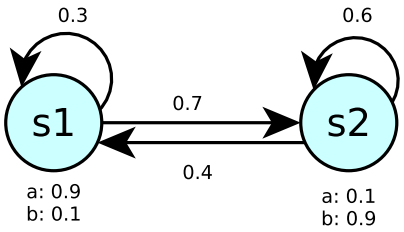

Figure 4.3: Graph representation of a simple HMM. Two states s_1 (s1) and s_2 (s2) are represented as nodes of the graph. Each state has its observation probability distribution over symbols a and b written below. The edges show all the possible transitions of states with their associated transition probabilities.

4.2.3 Homology Search Based on Profile Hidden Markov Model

Another family of methods of homology search is based on a statistical model called the Hidden Markov Model (HMM). The HMM models a system, which can be described as being in one discrete state from its set of states at every time instant [48]. Naturally, the state of the system changes over time. However, the state transition is not deterministic and is governed by a probability distribution over the set of its states. The exact shape of this distribution must be dependent only on the current state of the system. Otherwise, the system would violate the *Markov property* and would not be describable by any Markov model [49]. The choice of the starting state is governed by the initial probability distribution over the set of all states.

An outside observer is not able to perceive the exact state of the system at any time [49, 48]. The only things, which are visible, are system's observations. Observations are symbols from the observation alphabet of the system, which can be emitted to the outside world. This emission is not deterministic and, like in the case of the state transition, it is governed by a probability distribution over the alphabet of observation symbols. The emission can happen once after every state transition, or, in the case of a *null state*, it may not happen at all [27]. The whole system can be represented in a graph form as illustrated on the figure 4.3.

Formally, the HMM can be defined as a 5-tuple

$$(Q, V, A, B, \pi)$$

where $Q = \{q_1, q_2, \ldots, q_n\}$ is the set of states, $V = \{v_1, v_2, \ldots, v_n\}$ is the set of possible observations, $A = \{a_{ij}\}$; $a_{ij} = Pr(q_j, \text{ at t} + 1 \mid q_i \text{ at t})$ is the transition probability distribution over the set of states for a transition from the state q_i, $B = \{b_i(k)\}$; $b_i(k) = Pr(v_k \text{ at t} \mid q_i \text{ at t})$ is the probability distribution over the set of observations for the state q_i and $\pi = \{\pi_i\}$; $Pr(q_i \text{ at t} = 1)$ is the initial probability distribution over the set of states [48].

Model of a system in the format of HMM can be used to answer a number of problems. Typical inferential tasks, that can be solved using an HMM, include questions like: "Given a sequence of observations, what is the most likely sequence of states the system went through ?" or "How likely it is, that a given sequence was generated by this HMM ?" [48, 49].

The most common HMM model used in bioinformatics is called the profile HMM [27]. It is a special case of a linear HMM, which is typically constructed from a multiple sequence alignment, and in which the time domain is represented by a position in the sequence [27].

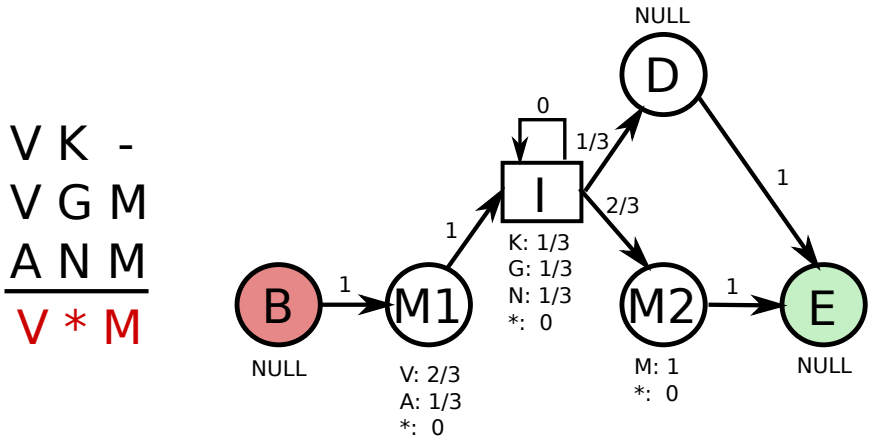

Figure 4.4: An example of a profile HMM created from a multiple sequence alignment. States B and E represent the beginning state and ending state of the HMM, M1 and M2 are match states, I is an insert state and D is a delete state. Emission probabilities are shown below each state. Transition probabilities are shown above edges. The word NULL denotes a state with no emissions. The multiple sequence alignment which served as the base for this example is shown on the left side.

The profile HMM model has three basic types of states: match states, insert states and deletion states. While match states represent the most conserved columns in a multiple sequence alignment, insert states represent less conserved regions, which may vary between proteins of the same family. The last group of states, deletion states, model the situation, when some of the sequences in the multiple sequence alignment have a gap at a highly conserved position. Apart from these, there is exactly one beginning state and one end state with no emission [27].

The emission and state transition probabilities are estimated by analyzing frequencies of residues in the multiple sequence alignment. For instance, if a given highly conserved position in the multiple sequence alignment contains valine (V) in 5 of 6 sequences and methionine (M) in one sequence, the emission probability of V in the corresponding state will be $5/6 = 0.83$. Similarly, the probability of M will be $1/6 = 0.17$. After the model was constructed, the alignment can be performed using the Viterbi algorithm and the alignment score can be calculated using the Forward algorithm [27]. Detailed description of these algorithms is beyond the scope of this master thesis. For further details, please refer to the article by Rabiner and Juang [48].

As in the case of a general-purpose HMM, the profile HMM can be represented in the form of a graph. Example of this representation is shown on the figure 4.4.

Chapter 5

Design of the System for Detection of Enzymes in Metagenomic Data

The primary goal of my master thesis is to create a set of tools for detection of enzymes in metagenomic data. However, the toolset should not detect arbitrary enzymes, but the ones, which are homologous to a query enzyme provided by the user. In this way, the tool allows to search for new enzymes, which have the same function, as some known enzyme. While the function may be the same, the found homologue can have better chemical and physical properties and can be interesting for protein engineering. In order to meet this goal, the toolset must be able to perform three main functions: *read pre-processing*, *search for homologues* and *enzymatic function verification* (figure 5.1).

The proposed toolset should work directly with the output of sequencing and should not require data to be pre-processed in any way. However, if the pre-processing was to be completely ignored, sequencing errors contained within the data may significantly decrease the quality of further analysis [38]. Because of this, the first task of the toolset is to pre-process the data with regards to its quality and output it in a format suitable for searching. I will refer to this format as a *metagenomic database* in the rest of my master's thesis. Further details of this process are described in the section 5.1.

After the pre-processing, the data can be used for detection. The detection is conducted using a protein sequence or a multiple sequence alignment provided by the user and its result is a set of homologues of the input sequence (multiple sequence alignment) contained within the metagenomic sample. The additional option of using multiple sequence alignment allows user to search for new enzymes which belong to the same enzymatic class as enzymes form the alignment. Moreover, the use of multiple sequence alignment emphasizes conserved regions and therefore allows to search for more distantly related enzymes [11]. Details of the searching method are the subject of the section 5.2.

While enzymes found by homology search have at least some level of structural similarity to the input query enzyme, they may still perform a different catalytic function. Therefore, in the last step, the toolset uses enzyme function verification methods to check whether found homologues have the same catalytic function as the input enzyme. Since the catalytic function of an enzyme is strongly depended on the configuration of its catalytic residues, this process requires user to provide information about their position on the input enzyme in the form of *catalytic region* annotation. Its format and details of proposed enzymatic verification methods are the subject of the section 5.3.

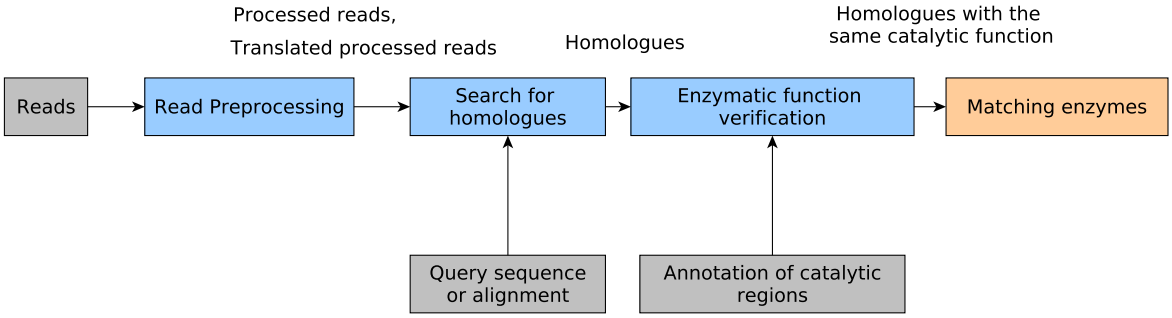

Figure 5.1: The architecture of the toolset for enzyme detection in metagenomic data. Inputs of the tool, shown in gray, are metagenomic reads, an input sequence or multiple alignment and an annotation of catalytic regions. The main functions of the toolset — data pre-processing, homology search and enzymatic function verification — are shown in blue. The orange box represents the final output of this toolset – homologous enzymes with the same catalytic function as the input query.

5.1 Metagenomic Read Pre-Processing

A set of reads, which is the result of a sequencing process, is not guaranteed to be completely error-free. Luckily, modern sequencing methods are able to provide quality estimation (i.e. belief of correctness) for every base in sequenced data. The pre-processing module exploits this information and transforms the set of reads into a searchable metagenomic database. The input file may be provided directly by the user, as a result of his own sequencing effort, or it might be downloaded from a public database, such as the NCBI Sequence Read Archive [29]. Its format should be compliant with the fastq format, which is a de facto standard for the storage of read sequences and their quality estimates [13]. Most metagenomic studies I have encountered were using paired read layout (reverse and forward reads); therefore I have decided to design the pre-processing tool primarily with respect to this type of data layout.

The first goal of the pre-processing stage is to remove regions within reads, which have insufficient quality, by a process called *trimming*. If the low-quality regions would be ignored, the result could be detrimental for further analysis and may lead to false interpretations of data [14]. In my design, I have decided to use a sliding window trimming approach, which was invented by Bolger, Lohse and Usadel [9].

During the trimming process, a window of constant size moves base by base from the 5' to the 3' end of every read (figure 5.2). In each position, the average quality of bases contained within the window is calculated. If the result is below a set quality threshold, the whole sequence from the beginning of the window to the end of the read is removed. If the removed portion is too long, the read can become too short to be useful in any further analysis. In order to keep the data as clean as possible, all reads with the length smaller then a set threshold are discarded. In the case of paired reads, this can result in some reads losing their partner from pair and become *unpaired reads*.

Nevertheless, low quality regions are not the only type of sequence, that should be removed. Many of the Next Generation Sequencing methods use special, short, artificially produced sequences, called *technical sequences*, to support the sequencing process. However, their presence could negatively affect the quality of subsequent analysis, because they were

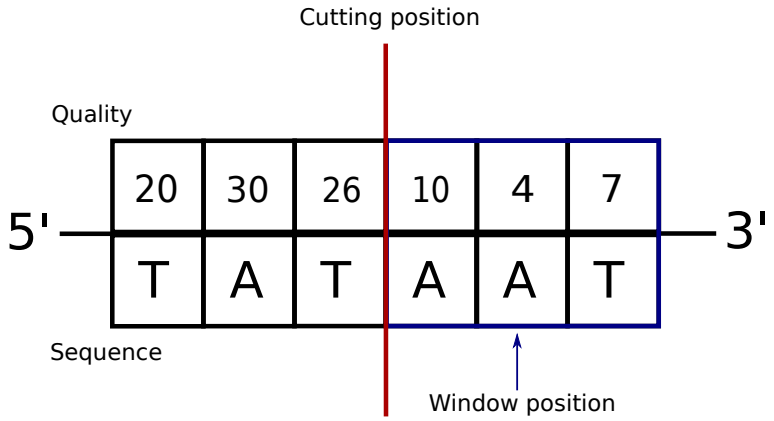

Figure 5.2: Trimming approach using a sliding window. The window (blue frame) advances from the 5' end to 3' end of the current read. If the average quality of bases in window falls below a set threshold, the whole sequence from the beginning of the window to the end of the read is removed. In this case the threshold was set to 10 and the average quality in current window is $(10+4+7)/3 = 21/3 = 7$. As a result, the read is cut at current position (denoted as "Cutting position"). The method was taken from paper by Bolger et al. [9].

not originally contained within the sample [9]. I have decided to use two techniques of their removal described in paper by Bolger, Lohse and Usadel [9] – the general method[1] and the palindromic method. While the first one is usable with any type of data layout, the second one is specific to the paired read data. The specificity of palindromic method provides higher accuracy for the paired data layout [9].

In general mode (figure 5.3), a known technical sequence is divided into a set of smaller substrings called seeds by the same windowing algorithm, as in the case of k-mers in the De Bruijn graph assembly method (subject of section 4.1.1). Reads are scanned from the 5' end to the 3' end and at each position, all seeds are compared with the current region. This comparison is not completely exact and allows a set number of mismatches. If any of the seeds matches, the match is extended using a local alignment algorithm and yields a local alignment score. Finally, if the alignment score exceeds a set threshold, the sequence is considered to be technical, and the read is cut from the beginning of the aligned region to the end of the whole read.

The palindromic mode (figure 5.4) is designed to remove short technical sequences, called adapters, which can appear at the end of a read [38]. Normally, these should not be present in data, however, if a read length is longer than a length of some of the DNA fragments, the reading process can go past the valid data into the adapter region. This problem is commonly referred to as the *adapter read-through* [9]. Since in the paired sequencing, each fragment is sequenced two times using both strands of the DNA, the adapter read-through can be easily detected. If it have happened, valid data regions of both reads in pair will be reverse complementary, equal in length, and both will end with a technical sequence [9].

In the first step of the algorithm in palindromic mode, both reads in pair are prepended with their respective adapter sequences. Then, reads are aligned, so that the prepended adapter of the second read follows immediately after the adapter of the first read. Afterwards, the overlap between the reads is continually extended and at each extension, a global alignment score is calculated. If the score is high enough, the reads are considered to be

[1] In the original article referred to as the simple method.

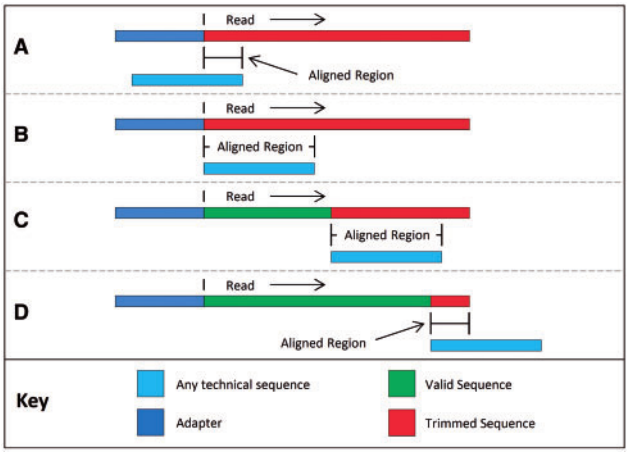

Figure 5.3: In the general mode, reads are scanned from the 5' end to the 3' end base by base and at every position an alignment score is used to determine whether a technical sequence is present. If it is, a region beginning from the start of the scanning window to the end of the whole read is cut (depicted in red). Situations in A, B, C, and D show different cutoffs based on a position of the match. The image was taken from paper by Bolger et al. [9].

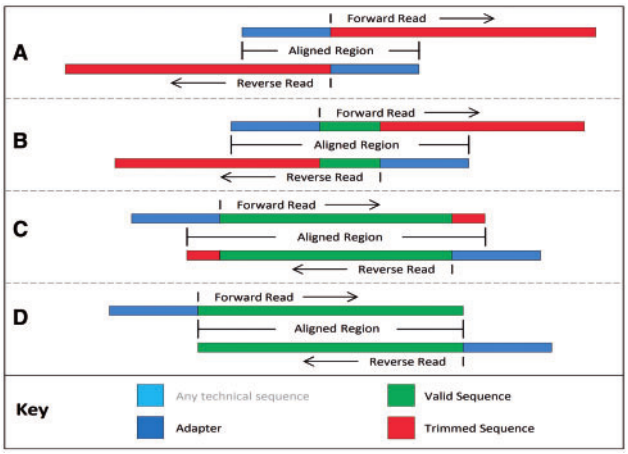

Figure 5.4: In the palindromic mode, reads in a pair are prepended with their adapters and aligned using a global alignment algorithm with iteratively increasing overlap. If the alignment score raises above a set threshold, ends of both reads are cut accordingly. Diagrams A, B, C, and D show situation for different length of overlap. The image was taken from paper by Bolger et al. [9].

aligned and each read is trimmed from the first base aligned with the prepended adapter of the other read.

In the final step of the pre-processing, unpaired reverse reads are reversed, complemented and merged with unpaired forward reads into a set of single reads. Sets of single reads, paired forward reads and paired reverse reads are translated into amino acid sequences using all six possible reading frames and results are merged into a set of protein sequences.

To summarize, the output of the pre-processing step is the set of protein sequences, the set of single reads, the set of paired forward reads and the set of paired reverse reads.

5.2 Search for Homologous Sequences

After pre-processing, the database is ready for searching. The search process expects either a query sequence or a multiple alignment of sequences. The first task of the search tool is to find homologues to the provided query in the metagenomic database.

Even though there is a number of homology search algorithms, I have decided to use the method based on the profile Hidden Markov Model (profile HMM) as the primary search method, because it is able to work with multiple sequence alignments directly [27].

In the first step, an alignment or a sequence is converted into a corresponding HMM. The obtained model is than used to search through the set of protein sequences (translated reads). Since it is very unlikely to find a whole enzyme in one short read, the search is

limited only to protein domains of the input query. The result is a set of translated reads, in which some parts resemble the sequence of some domain from the input enzyme.

In the next step, untranslated versions of matching reads are extracted from the set of paired forward reads, paired reverse reads and single reads. These are then assembled into longer contigs using the assembly method based on the De Bruijn graph. Contigs can be much longer than plain New Generation Sequencing reads, and therefore the chance of finding a whole enzyme in one translated contig is significantly higher.

Finally, all contigs are translated into amino acid sequences using all six reading frames and are subjected to another search using the same profile HMM as was used in the search for protein domains. However, this time the search is not limited to protein domains and tries to find the whole enzyme. Matching sequences are the resulting output of the search pipeline.

5.3 Enzymatic Function Verification

Enzymes outputted from the homology search are not always true homologues to the provided search query. For instance, a found enzyme may be missing some small region, which is not long enough to make a difference in homology search, yet it can be critical for its catalytic function. Even if the found enzyme is complete, the result could still be incorrect. This is due to the fact, that enzymes with very similar structure (and usually similar sequence) may perform very different function [45].

In order to address this situation, I have decided to incorporate enzymatic function verification as the last filtering step of the processing pipeline. In this step, the user provides an annotation of *catalytic regions* on the query enzyme. By the term catalytic region, I will refer to an area on the sequence of enzyme, which contains one or more catalytic residues following immediately after each other. Their annotation has to contain a position of the beginning and end of each region. More formally, the annotation can be defined as a set $A = \{(b_0, e_0), (b_1, e_1), \ldots, (b_n, e_n)\}$, where b_n is a position of the first catalytic residue of the region, and e_n is a position of the first residue following immediately after the region. Program uses this annotation to verify enzymatic function of found enzymes. In following sections, I propose a method of verification based on a normalized cross-correlation and a group of methods based on an alignment.

For the sake of clarity, in the following descriptions, I will refer to the enzyme, which was found in homology search and its function is to be verified as an *enzyme candidate*. The enzyme, which serves as the base for verification and whose catalytic regions are given in the annotation will be referred to as a *query enzyme*.

5.3.1 Methods Based on Alignment

All of the alignment methods follow the same basic algorithmic scheme presented in the algorithm 1. This scheme can be divided into three main steps - *alignment*, *offset array calculation* and *iterative scoring*. The input common to all of the proposed methods is the set of catalytic region annotations A, the sequence of an enzyme candidate S_1 and a sequence of a query enzyme S_2. In case of multiple sequence alignment, the user can provide its consensus sequence, or a sequence of arbitrary enzyme from the alignment in place of the input S_2.

Firstly, each of these methods starts with an alignment of the sequence of the enzyme candidate with the sequence of the query enzyme. This is the purpose of the function

align, which takes two strings S_1 and S_2 over the standard IUPAC alphabet and produces two aligned strings G_1 and G_2 over the alphabet $IUPAC \cup \{-\}$, where the character "–" represents a gap in the alignment. I have decided to use Needleman-Wunch global alignment algorithm, however, it is possible to use different method. Strings G_1 and G_2 have to have the same length, and all gaps resulting from the alignment should be denoted by the gap character.

input : $A = \{(b_0, e_0), (b_1, e_1), \ldots, (b_n, e_n)\}$, String S_1, String S_2, Scoring matrix M, Catalytic region weight w_a
output: p – score of the verification; value from interval $[0, 1]$

```
p = 0;
i = 0;
(G1, G2) = align(S1,S2);
arr = offsetArray(G2);
for (b,e) in A do
    b = b + arr[b];
    e = e + arr[b];
    (W1, W2) = window(E1,E2,b,e,sw);
    p += score(W1, W2, b, e, M, wa) ;
    i++;
end
p = p/i;
```

Algorithm 1: Basic algorithmic scheme of methods based on alignment. In the first step, both, the sequence of an enzyme candidate and the sequence of a query enzyme, are aligned. Then, a helper data structure, called offset array, is created and used for adjustment of catalytic region positions in the aligned query enzyme sequence. The algorithm iterates over all catalytic regions of the query enzyme and uses windowing and scoring to estimate likelihood that the input sequences S_1 and S_2 belong to the same family of enzymes. Resulting scores are summed and divided by the number of catalytic regions.

The insertion of gaps into the query enzyme sequence, which frequently happens during the alignment process, can change the positions of its catalytic regions. Because of this, the coordinates of catalytic regions have to be adjusted. This is the purpose of a helper data structure which I will refer to as the *offset array*.

Given a string over the alphabet $IUPAC \cup \{-\}$, the value of the offset array at index i represents the number of gaps ("–" symbols) between the beginning of this string and i-th non-gap symbol (valid IUPAC symbol). For example, given a string "M--I-SSISIPI" with starting index 1, the value of the offset array at the index 2 is 2, because the second IUPAC symbol "I" is preceded by two gaps. Analogically, the value at index 3 is 3, because the third IUPAC symbol "S" is preceded by three gaps.

In general, if there was a symbol in the query sequence at some position j and the alignment algorithm inserted some gaps on preceding positions, the position of the symbol in the aligned version will be advanced by the number of gaps inserted. More formally, the new position will be equal to $j' = j + n$, where n represents the number of preceding gaps. The number n is equal to the value of the offset array at the index n, and therefore the offset array can be used to adjust the beginning and ending position of a catalytic region.

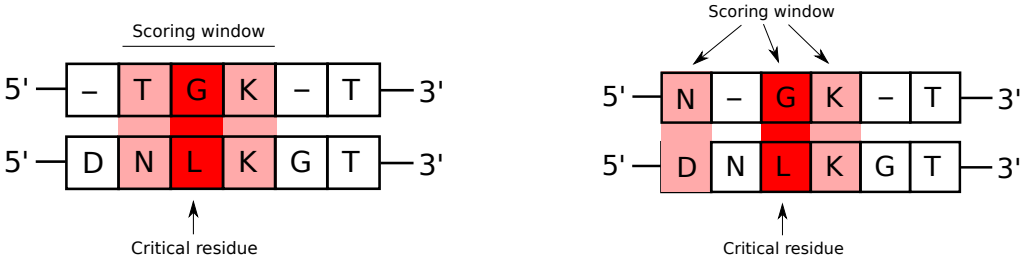

Figure 5.5: Two proposed windowing strategies – the *fixed window* and the *soft window* approach. In the first one, catalytic regions (bright red) and n amino acids on both sides of them (dim red) are included into the window. In the second approach, positions where any of sequences contains the gap character are omitted.

After the adjustment, substrings containing catalytic regions are extracted from the query enzyme. Apart from the catalytic region itself, a defined number of extra symbols from both sides of it is included into every substring. I will call this extraction process windowing, and I propose two strategies of it – the *fixed window method* and the *soft window method*.

In the first, straightforward approach, the catalytic region and a fixed number of neighboring symbols is included into the substring. In the second approach, only positions where both, the candidate sequence and the query sequence, contain valid amino acid symbol are included. The difference is illustrated on the figure 5.5. Substring corresponding to the same window is also extracted from the candidate enzyme sequence. These two substrings are the input of all scoring methods.

Finally, after the corresponding regions have been extracted in the windowing process, the scoring function (in the algorithm 1 named *score*) calculates a value between 0 and 1, which represents the likelihood of the query enzyme and the candidate enzyme having a same catalytic function. I propose three scoring methods: *simple scoring method*, *matrix scoring method* and *weighted matrix scoring method*.

The simple scoring method compares residues between substrings one by one and divides the number of matching ones by the total length of substrings[2]. In effect, the result is a ratio of matches to the number of all symbol comparisons.

This method can be easily enhanced by the use of a scoring matrix. In this extension, called the *matrix scoring* method, residues are compared one by one like in the previous case, but instead of a match count, values from the scoring matrix corresponding to each compared pair are summed up. Having the sum of scores s, the resulting score p is calculated as

$$p = \frac{s}{nm}$$

where n is the length of the window and m is the maximal value from the scoring matrix. In effect, the preceding term calculates the ratio of obtained score to the maximal score, that could have been obtained by the use of particular scoring matrix.

The catalytic function of enzymes strongly depends on the configuration of their active sites. This effect can be reflected in the scoring approach by giving the catalytic region higher importance during comparison. This is the purpose of the weighted matrix scoring method. Its scoring algorithm is exactly the same, like in the case of the matrix method, but while the scores from the outside of the catalytic region are added to the sum directly,

[2]Both substrings have the same length.

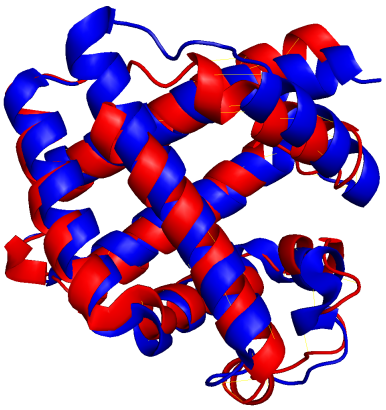

Figure 5.6: Alignment of a sperm whale (Physeter macrocephalus; shown in blue) myoglobin and leghaemoglobin from Lupinus luteus (shown in red). While both of them are responsible for oxygen transport and are very similar in structure, they yield only 19.46% sequence similarity. The figure is based on an example from an article by Gao and Li [18].

the scores of residues within the catalytic region are multiplied by a weighting coefficient $w \geq 1$. Formally, this results in the equation

$$p = \frac{s}{n_0 m + n_1 m w}$$

where n_0 is the number of residues outside the catalytic region and n_1 is the number of residues contained directly within the catalytic region.

As the final adjustment, the score p can be multiplied by a partial score of the catalytic region in order to further highlight its importance. The equation in this case becomes:

$$p = \left(\frac{s}{n_0 m + n_1 m w} \right) s_a$$

where s_a is the score of a catalytic region.

Finally, the scores of all catalytic regions are summed and the result is divided by the total number of regions on the query enzyme. The resulting value is compared with a predefined threshold, and if it is higher, the candidate enzyme is considered to have the same catalytic function as the query enzyme.

5.3.2 Method Based on Normalized Cross-Correlation

While it is commonly believed, that structurally similar proteins have also very similar sequence, recent studies provide examples of proteins, which are structurally very similar, yet their sequences are very different (figure 5.6 illustrates an example of this phenomenon) [18, 2]. If this difference is in residues belonging to some active site, alignment verification methods from the preceding section may not work correctly.

This problem can be solved by the use of a better descriptor for amino acids, which would provide more information about the residue than a plain symbol does. One of the possible descriptors was proposed in article by Kidera et al. [25]. In their research, authors have measured 10 different chemical and physical properties of each of the 20 naturally occurring amino acids. As a result, every amino acid can be represented as a vector of 10 dimensions, where every dimension corresponds to one chemical or physical property, and the whole sequence can be expressed as a matrix (table 5.1). If every amino acid from the sequence corresponds to one column, the resulting matrix will have 10 rows and number of columns equal to the length of the amino acid sequence.

The matrix representation of a sequence allows use of different comparison methods. One such method, called normalized cross-correlation, was used for protein structure comparison by He et al. [20]. Originally, this method was developed for applications in computer vision and first introduced in article by JP Lewis [30]. Given an image f and a template t, the normalized cross-correlation $corr$ of template t and image f at coordinates u and v is equal to:

$$corr(u,v) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})(t(x-u,y-v) - \bar{t})}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}(t(x-u,y-v) - \bar{t})^2}}$$

where $\bar{f}_{u,v}$ is an average intensity value in the region of the image f, which is centered at the coordinates u, v and has equal dimensions as the template t, and the symbol $\bar{t}$ denotes average intensity value of the whole template t. The result is a matrix of correlation coefficients.

He et al. [20] took this equation and applied it to the case of the amino acid matrices described before. As it is logical to compare only the same physical and chemical properties, the calculation of correlation with vertical displacement can be completely neglected [20]. With this in mind, the equation can be adjusted into the following form:

$$corr(u) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u})(t(x-u,y) - \bar{t})}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u})^2 \sum_{x,y}(t(x-u,y) - \bar{t})^2}}$$

Unlike the case of general normalized cross-correlation, the output of the method for amino acid matrices is only a correlation vector [20].

In my approach, I use the normalized cross-correlation proposed by He et al. [20] for enzymatic function verification. Following is the description of the main steps of this method.

Firstly, for each catalytic region on the query enzyme, a substring is extracted using the fixed window method described in the section 5.3.1. This substring is correlated with the sequence of the candidate enzyme using the normalized cross-correlation. The maximal value of correlation vector is used as the individual score of the catalytic region. The same steps are repeated for every catalytic region on the query enzyme.

After the scoring, all results are summed up and divided by the number of catalytic regions on the query enzyme. Like in the case of methods based on alignment, the resulting score is compared with a predefined threshold, and if it is higher, the candidate enzyme is considered to have the same catalytic function as the query enzyme.

S	W	T	W	E	N
0.81	0.30	0.26	0.30	–1.45	1.14
–1.08	2.10	–0.70	2.10	0.19	–0.07
0.16	–0.72	1.21	–0.72	–1.61	–0.12
0.42	–1.57	0.63	–1.57	1.17	0.81
–0.21	–1.16	–0.10	–1.16	–1.31	0.18
–0.43	0.57	0.21	0.57	0.40	0.37
–1.89	–0.48	0.24	–0.48	0.04	–0.09
–1.15	–0.40	–1.15	–0.40	0.38	1.23
–0.97	–2.30	–0.56	–2.30	–0.35	1.10
–0.23	–0.60	0.19	–0.60	–0.12	–1.73

Table 5.1: Example of a sequence translated into a matrix form using the method proposed by Kidera et al. [25]. Each amino acid is represented by a vector of 10 dimensions. As a result, the whole sequence is represented as a matrix where each residue is described by one column. Figure is a modified version of one presented in the paper by He et al. [20].

Chapter 6

Implementation of the Proposed System

The system proposed in the preceding chapter was successfully implemented using the Python 3, C and Bourne Shell programming languages. Its implementation consists of three main modules that correspond to the three main parts of the system presented in the specification – pre-processing, homology search and enzymatic function verification. All of their implementation files are located in the `src` subdirectory on the attached CD. Further details about contents of the CD are presented in the appendix A.

The pre-processing module, described in the section 6.1, takes raw metagenomic reads and outputs them in a searchable format that I refer to as the *metagenomic database*. Core of its implementation lies in scripts `process.sh` and `dispatcher.sh` located in the `src/metacentrum` directory.

The second module, which is the subject of section 6.2, uses metagenomic database to search for novel enzymes based on a search query. Both, the search module and the pre-processing module, are written mainly in the Bourne Shell programming language, and are able to function on a personal computer with Linux system, as well as, on a grid computing environment with PBS Pro scheduler. Like in the case of pre-processing, search scripts, `searchdb.sh` and `metasearch.sh`, are located in the `src/metacentrum` directory.

The third module, described in section 6.3, is mostly written in Python 3, and implements all proposed enzymatic verification methods. Together, with other utility and supporting functions, it is part of the `biodb` Python 3 library located in the directory `src/biodb`.

The following sections provide further technical details about the implementation of the proposed system. For information about its usage, please refer to the tutorial located in the appendix E.

6.1 Metagenomic Read Pre-Processing

The pre-processing pipeline is implemented in the script `process.sh` that is located in the directory `src/metacentrum`. When executed, this script expects two fastq files with raw reads and their qualities as its input – a file with forward reads and a file with reverse reads. If these are present, the script continues by applying a number of pre-processing operations. An overview of the whole process is depicted on the figure 6.1.

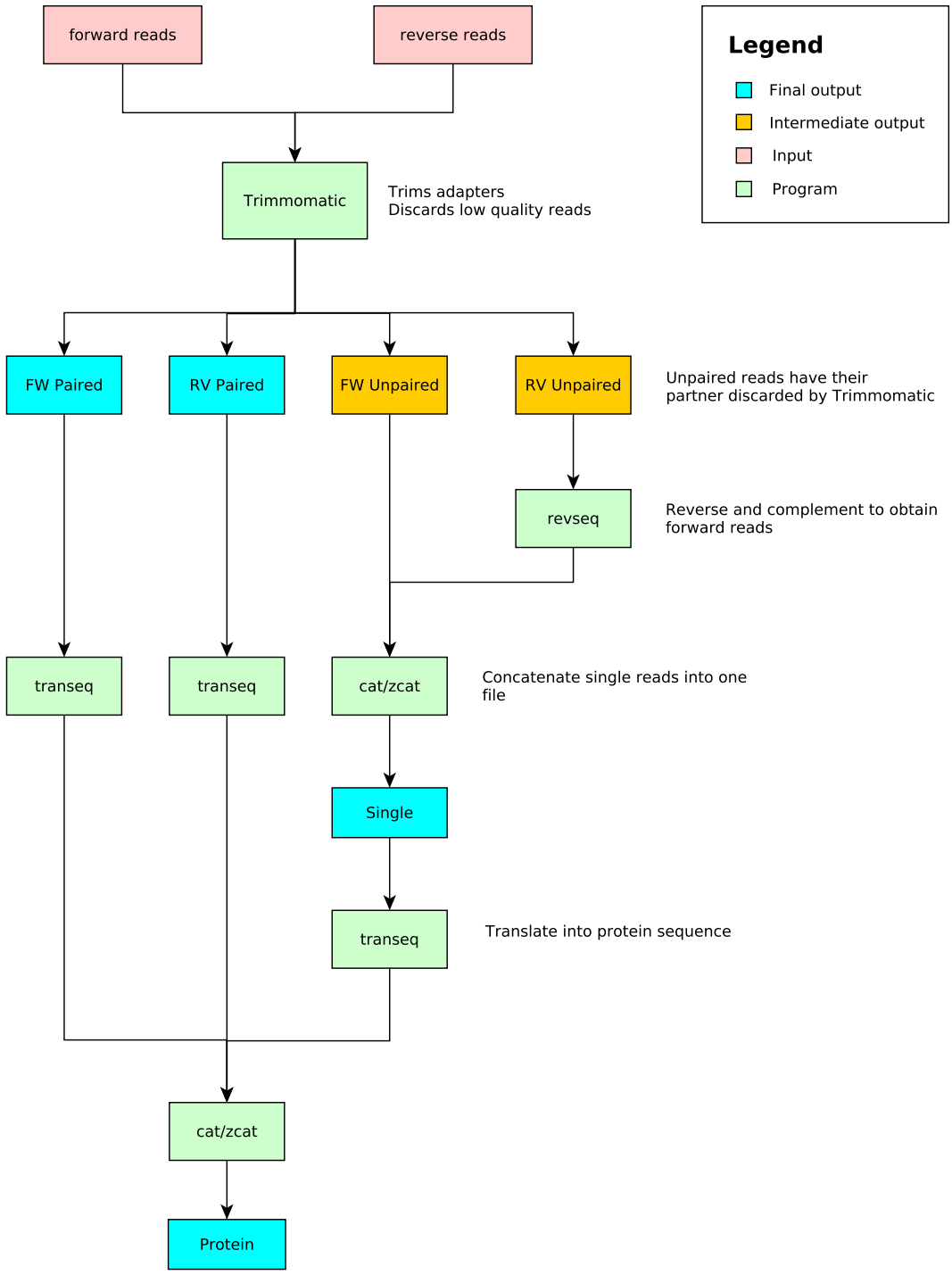

Figure 6.1: The pre-processing pipeline applies series of pre-processing operations to files with forward and reverse reads. Most important of these is low-quality region cutting and technical sequence removal provided by the program Trimmomatic [9]. Reads, which lost their partner from pair, during the read cutting, are merged into a single fasta file. After pre-processing, all reads are translated into protein sequences by the program transeq from the European Molecular Biology Open Software Suite [50]. The output of pipeline consists of processed read files and of a file containing translated versions of all reads.

The main part of quality pre-processing is provided by an external program, called Trimmomatic, that was presented in paper by Bolger, et al. [9]. This program removes low-quality regions from reads by combining three cutting methods – leading base cutting method, trailing base cutting method and cutting method based on a sliding window. The leading and trailing base cutting methods remove bases incrementally from both ends of reads, until a base with sufficient quality is encountered. The sliding window method is more complex and instead of processing bases one by one, uses quality value averaging over a sliding window. Further details about this method are provided in the section 5.1.

In order to determine whether a region has sufficient quality, its quality value must be compared with some threshold. This threshold can be different for every cutting method, and is specified as a command line argument of Trimmomatic. In order to give the user sufficient control over the pre-processing process, these thresholds are also specified in command line arguments of the script `process.sh`.

Apart from low-quality region removal, Trimmomatic is also able to remove regions containing technical sequences. These artificial sequences are added into the sample during the sequencing process and their presence could lead to errors in analysis. For this purpose, the program provides two methods – simple method and palindromic method. Both of these identify technical sequences in reads by comparison with their standard form. Standard forms of technical sequences are different for every sequencing technology and are provided by its manufacturer. Since these are not included directly in the Trimmomatic package, the user is required to supply them in a fasta file[1]. Both, palindromic method and simple method, are described further in the section 5.1.

The output of Trimmomatic consists of four fastq files – forward paired reads, reverse paired reads, forward unpaired reads and reverse unpaired reads. The two unpaired files contain reads, whose partners had very low quality, and were discarded during the cutting process completely.

After the pre-processing by Trimmomatic, the reverse unpaired reads are reversed and complemented using the revseq program from the European Molecular Biology Open Software Suite (EMBOSS) [50]. This allows reverse and forward unpaired reads to be merged into a single fastq file. At this point, the pre-processing with regards to quality is completed, and in order to save disk space, all fastq files are converted into corresponding fasta files. The conversion is provided by the program seqret from the EMBOSS suite.

Finally, in the last step, all fasta files are translated using the program transeq, which is also part of the EMBOSS package. The translation is conducted for each of six possible reading frames and results in six translated records per every read. Translated versions of all read files are concatenated and stored in a single fasta file (protein.fasta).

The final output of the processing script consists of four files – forward reads (out_1.fasta), reverse reads (out_2.fasta), single reads (single.fasta) and translated reads (protein.fasta).

Since, in practice, the size of a metagenomic sample can reach hundreds of gigabytes, a typical personal computer may have insufficient power for its effective processing, and the execution can take a long time. In order to address this situation, I have decided to use MetaCentrum grid computing infrastructure, which can accelerate the task by providing massive parallelization.

The implementation for MetaCentrum is located in the wrapper script `dispatcher.sh` that is built on top of the script `process.sh`. While the script was tested on MetaCentrum only, it should work on any grid system with PBS Pro task scheduler installed. Like the

[1] With the exception of standard Illumina TruSeq3 adapter sequence. Right for its distribution was granted to authors of Trimmomatic by its owner – company Illumina [9].

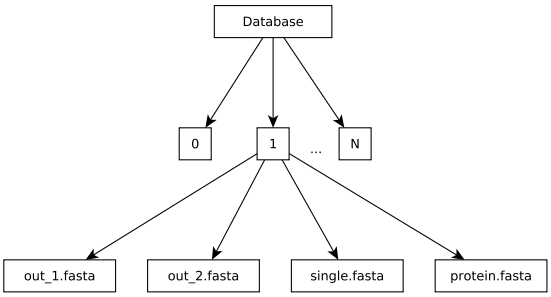

Figure 6.2: Resulting directory layout after processing on a grid computing infrastructure using the script `dispatcher.sh`. Each subdirectory contains a block of data, which was processed by one node. While, the data could be merged into a single set of files, the chunked layout is preserved, so that the parallelized searching script does not have to split the files again.

script `process.sh`, the dispatcher is written in Bash and has the same main inputs – the forward and reverse read fastq file. Apart from providing these files, the user is also required to specify parameters of parallelization – the number of processing nodes and the number of processors to use per node.

When executed, the dispatcher dynamically generates and schedules short scripts to be executed on processing nodes (one per each node). Each generated script copies chunk of both input fastq files onto its processing node. Given N nodes numbered $1, \ldots, N$ and a fastq file with M entries, each node will receive M/N records. Furthermore, input files are split and copied in an interleaved fashion. This means that the node 1 receives every 1-st record, node 2 every 2-nd record, and so on up to the M-th node, which receives every M-th record. Chunking is not implemented directly in the dispatcher script, but an external program `filter_fastq.py` written in Python 3 is used.

After the chunking and copying, each generated script runs the `process.sh` script on its block of input data. Quality thresholds required for the execution are extracted from a quality configuration JSON file, which has to be provided by the user. Apart from thresholds, this file has to contain the name of a fasta file with technical sequences for sequencing technology, which produced input metagenomic reads. Its example with description is presented in the appendix B.

After the processing, all files with partial results are copied back from processing nodes into the directory, from which the dispatcher script was executed. It is important to note, that files from nodes are not merged and are left in a split form. Therefore, the final result of the dispatcher script consists of M directories named $0, 1, \ldots, M-1$, each containing forward read file, reverse read file, single read file and a file with translated reads (diagram of this directory structure is presented on the figure 6.2). In further sections, I will refer to the output of dispatcher script as a *metagenomic database*.

6.2 Search for Homologous Sequences

The whole searching process, depicted on the figure 6.3, can be divided into three main steps: homology search, assembly and enzymatic function verification.

The first step of the searching process, homology search, is implemented in the script `searchdb.sh`. When executed, the script uses file with a search query to search through the metagenomic database for matching enzyme candidates. The search query should either be a fasta file with a single protein sequence or a multiple sequence alignment. However, in the latter case, the alignment file must be converted into a profile hidden markov model

and stored in a format compatible with software HMMER[2] [16]. Apart from the search query, the script also requires pre-processed metagenomic reads (paired, forward, single and translated read file).

The homology search implementation supports both methods described in the section 4.2 – the BLAST heuristic and homology search based on a Profile Hidden Markov Model. User can select the desired method through command line arguments of the script `searchdb.sh`. Even though these methods lie at the core of the search, they are not part of the script itself and are provided by external programs. For the BLAST heuristic, I have decided to use implementation from the National Center for Biotechnology Information (NCBI) [36]. For the option of Profile Hidden Markov Model, I have chosen the software HMMER [16]. While HMMER is able to use as its search query both, a single sequence and a profile HMM, BLAST is not capable to use HMM. Therefore, if an HMM is provided, HMMER is used regardless of the method selected by the user.

The search script uses one of these programs to search through the file with translated reads. The result is a set of translated reads, which have a significant similarity with the search query. In order to maximize the chance of finding a whole enzyme, these have to be assembled into longer sequences. Since, most modern assemblers require nucleotide sequences, corresponding untranslated reads have to be extracted from the forward, reverse and single read file using the utility `filter_fastq.py`. In case of paired reads, if translated sequence of one read from pair has a significant match, the other read is automatically extracted, regardless of whether its translated sequence is similar to the input query or not. Obtained reads are stored in fasta files that correspond to the input read files – matching forward reads, matching reverse reads and matching single reads file.

These files can be assembled into longer sequences (contigs) by assembly method based on a De Bruijn graph described in 4.1.1. I have chosen its implementation, called Velvet, proposed by D. R. Zerbino and E. Birney [58]. After a number of experiments, I came to conclusion, that I was able to achieve best assembly only by individual fine-tuning of assembly parameters for each set of homology search results. Therefore, I have not included the assembly step in the script `searchdb.sh` and the user is required to assemble reads manually.

After the assembly, the user can use the script `verify.sh` to apply one of the enzymatic function verification methods onto the set of assembled contigs. Since the verification is based on a comparison with some known enzyme, the user has to provide a JSON file containing its sequence and annotation of its catalytic regions (format of this file is described further in the appendix C). Apart from that, the script also requires the search query file used in homology search.

In the first step of its execution, the verification script uses program transeq to translate assembled contigs into protein sequences under all six reading frames. This step is crucial, because proposed enzymatic function verification methods can be only applied to protein sequences. However, not all sequences produced by translation can be valid enzyme candidates. For every gene, only translation using one of the six reading frames leads to a valid protein sequence. Furthermore, some of the sequences may have been wrongly assembled and all of their translations are inevitably invalid. In order to remove these erroneous sequences, the script repeats the search over translated contigs using BLAST or HMMER with the same search query sequence as the one used in the script `searchdb.sh`. Only sequences, which are results of the second search, are kept in the contig file. Afterwards,

[2]HMMER suite directly provides utility hmmbuild that is able to convert a multiple sequence alignment into an hmm file.

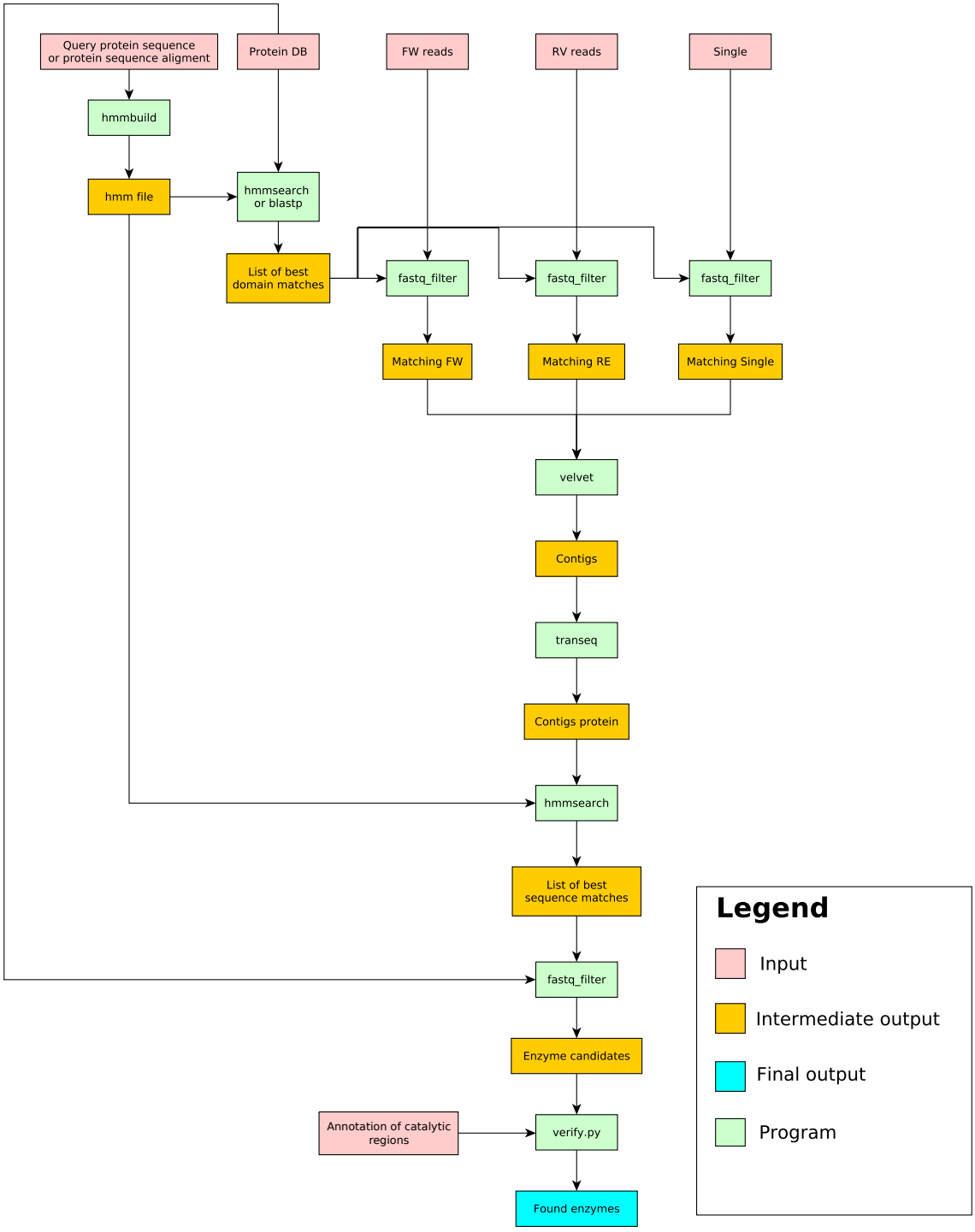

Figure 6.3: Diagram of the search system implementation. The input of the system consists of a file with translated reads, processed read files, file containing an input search query and a file with an enzyme sequence and annotation of its catalytic regions. First, the translated file is searched by either BLAST or HMMER. Next, untranslated reads corresponding to found sequences are extracted. Afterwards, the resulting set of reads must be manually assembled by the user. Finally, after the assembly, contigs are translated and filtered based on their sequence similarity using BLAST or HMMER and than using `verify.py`, in order to filter out enzymes, which lack the desired catalytic function.

an enzyme verification method is applied to assembled sequences, in order to discard those, which do not posses the desired catalytic function. The user can select any of proposed verification methods through the command line arguments of the verification script. Internally, the script propagates method setting and input files to the python program `verify.py`. The result of its execution is a file with identifiers of sequences, which posses the the desired catalytic function, along with their verification scores. Details of implementation of enzyme verification methods are subject of the next section.

As in the case of sample pre-processing, the speed of the search can be improved by parallelization using the MetaCentrum grid computing environment. For this purpose, I have implemented a wrapper script `metasearch.sh`. As its input, the script excepts a valid metagenomic database and a file with search query. When executed, it spawns a number of jobs that is equal to the number of chunks (subdirectories) of the metagenomic databse. Each of these jobs firstly copies one chunk from the database onto its processing node. Then, it executes homology search using the script `searchdb` and copies the search results back to the directory, from which the `metasearch.sh` script was executed. Finally, after all jobs have finished their execution, a collector job merges search results from all processing nodes. Results of the parallelized script are equal to results of the homology search script `searchdb.sh`. The subsequent assembly and enzymatic verification steps are not subject of parallelization. In all of my experiments, the results of homology search were small enough to be comfortably processed on a personal computer, and therefore, I have concluded, that the overhead of parallel processing would be greater than its benefit.

6.3 Enzymatic Function Verification

The main purpose of enzymatic function verification methods is to decide, whether two arbitrary enzymes have the same catalytic function. While one of them should be already known, the other is expected to be a novel enzyme candidate. The decision is based on their sequences and on the list of catalytic regions of the known enzyme. In essence, this problem can be viewed as a classification task, in which the enzyme candidate is to be classified into the class of enzymes with the same catalytic function as the known enzyme, or into the class of enzymes which do not posses this function.

The central part of the enzyme verification implementation is a hierarchy of classifier classes located in the source file `src/biodb/biodb/enzyme.pyx` (corresponding class diagram is depicted on the figure 6.4). The root of the hierarchy is an abstract class `Classifier` and all of the verification methods are implemented in its inherited classes. Namely, these classes are: `gAlignmentClassifier` (methods based on global alignment), `lAligmentClassifier` (methods based on local alignment) and `correlateClassifier` (method based on correlation). Each of these provides a method `call`, which given a known enzyme and an enzyme candidate with annotated catalytic regions, returns a classification score. Expected format of enzyme data structures is presented in the appendix C. Since each class implements different verification approach, their `call` methods differ as well. Descriptions of these variants are presented in following sections.

Apart from classifier hierarchy, file `enzyme.pyx` also contains class `classifierProvider`, whose purpose is to provide means to instantiate and configure classifier classes from a string descriptor. This descriptor should contain an identification of a particular classifier and parameters of its classification method and should be provided as a command line argument to the verification script `verify.sh`. Details about its syntax are provided in the appendix D.

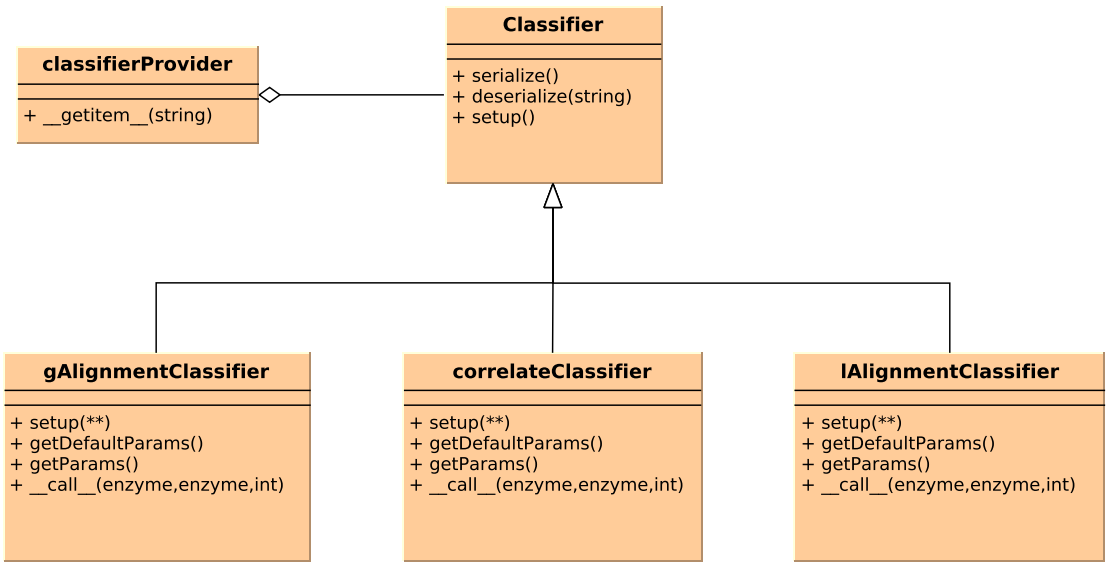

Figure 6.4: Figure illustrates the hierarchy of classifier classes. Its root is the class `Classifier`, which implements general serialization and deserialization operations. Its inherited classes provide implementations of enzyme verification methods. All of them must contain a method `call`, which returns a classification score for two provided enzymes. The class *classifierProvider* serves as a dispatcher responsible for instantiation of classifiers based on a descriptor string. Syntax of this string is the subject of appendix D.

6.3.1 Methods Based on Alignment

The implementation of methods based on global alignment follows the proposal presented in the section 5.3.1 and is located in the `call` method of the class `gAlignmentClassifier`. Its first step is an alignment of two input enzyme sequences. For this purpose, I have decided to use functions `align.globalxs` and `align.globald` from the `pairwise2` module of the Biopython Python 3 library [12]. While the function `align.globalxs` uses a simple scoring model with score 1 for match and 0 for mismatch, the function `align.globald` uses a scoring matrix. The choice of a scoring model, scoring matrix, as well as choice of other classification parameters is provided through the method `setup` of `gAlignmentClassifier`. However, the range of available scoring matrices is limited to those provided by the `MatrixInfo` module of the Biopython library.

After the alignment, a classification score of two aligned sequences is calculated using the function `score_global` defined in the file `scoring.pyx`, which is located in directory `src/biodb/biodb/util/`. Internally, `score_global` creates an offset array using the function `mkOffsetArr` and calls the function `score_simple` for each catalytic region. The function `score_simple` uses the offset array to adjust the beginning and ending position of each region, applies windowing and scoring and returns a classification score. While scoring is mainly implemented in this function, soft and fixed windowing strategies are implemented in functions `n_compare_direction` and `compare_fixed`. Like the function `score_global`, functions `mkOffsetArr`, `score_simple`, `n_compare_direction` and `compare_fixed` are also defined in the file `scoring.pyx`. The final classification score

returned by `score_global` is the average score across all catalytic regions of the known enzyme.

Apart from its use in alignment, a scoring matrix can be also used to calculate the enzymatic verification score. In such case, it must be provided as an argument to the call of the function `score_global`. If no explicit matrix is given, the function uses special matrix `idMatrix` defined in the file `enzyme.pyx`, which is equivalent to the simple scoring model (1 for match, 0 for mismatch).

Furthermore, the alignment can result in insertions of a special gap character. This character is not included in standard scoring matrices and its scoring is addressed by the function `score_simple` explicitly. The comparison of a gap character with a non-gap character has a value of 0 and the comparison of two gap characters has a value 1 regardless of the used matrix.

Apart from proposed verification methods, which use global alignment, I have decided to implement a set of simple methods based on local alignment. Their implementation is located in the class `lAlignmentClassifier`. Its `call` method uses function `score_local` defined in the file `scoring.pyx` in order to perform scoring. This function extracts each catalytic region and a fixed number of residues on both sides of it. Extracted fragments are aligned to the sequence of the enzyme candidate using the Smith-Waterman algorithm (function `align.localds` from the `pairwise2` module of the Biopython library). In contrast to global methods, no special scoring operation is applied and the resulting score for each region is the raw alignment score returned by the function `align.localds`. The final score is an average of scores across all catalytic regions.

During the evaluation phase of my research, in which a great number of enzyme sequences have been used, it became quickly apparent, that in order to achieve a reasonable execution speed, the critical operations of alignment and scoring must be as fast as possible. In order to achieve this goal, source files `scoring.pyx` and `enzyme.pyx` have been compiled using the compiler for Cython programming language. Instead of using the standard Python interpreter, Cython files are translated into the C programming language, compiled and than linked as dynamic libraries. In this way, module compiled by Cython can achieve a big performance improvements over a plain Python implementation [7].

6.3.2 Method Based on Normalized Cross-Correlation

The verification method based on the normalized cross-correlation is implemented in the `call` method of the classifier `correlateClassifier`. When executed, this method translates both enzyme sequences into a matrix form using the function `aaToMatrix`[3] located in the file `enzyme.pyx`. Afterwards, it iterates over catalytic regions of the known enzyme and extracts a submatrix containing the catalytic region and a fixed number of residues on both sides of it. Each submatrix is correlated with the whole matrix of the enzyme candidate using the function `norm_correlate` from the file `sequence_correlate.pyx` (located in the directory `src/biodb/biodb/util`). Output of this method is not a whole correlation matrix, but only its maximal value. Finally, the average of maximal correlation values across all catalytic regions is returned as the final classification score.

The function `norm_correlate` calculates correlation using the normalized cross-correlation method presented in the section 5.3.2. In order to achieve optimal performance, I have decided to implement it in the C programming language. In effect, the Python function

[3]The principle of this translation is described in the section 5.3.2.

`norm_correlate` serves only as an interface to this external implementation. The C code is located in files `ncorrelate.c` and `ncorrelate.h`.

Chapter 7

Evaluation of the Proposed System

After the system has been successfully implemented, it was neccessary to evaluate its performance and analyze its abilities. In order to do this effectively, I have decided to evaluate enzymatic verification module and homology search module separately. In both cases, datasets required for testing were obtained from public databases or generated *in silico*. The pre-processing part of the system was not subject of the evaluation, because all pre-processing strategies used are well-known, their use is widespread and have been previously tested [9].

7.1 Homology Search in Metagenomic Data

A suitable set of metagenomic samples is crucial for proper evaluation of the search module. These can be found in a number of online databases, such as the Sequence Read Archive (SRA) provided by the International Nucleotide Sequence Database Collaboration (INSDC) [29], which I have chosen as the primary source for my research. Since metagenomic samples often require a big amount of storage space, it is not possible to process them massively and it is neccessary to pick few suitable candidates.

After some consideration, I have decided to use two real samples from the SRA and one artificial, generated on a computer (for their summary, see table 7.1). The first real sample (NCBI accession SAMN00737776) was collected from prairie in Kansas, USA as a part of the Great Prairie Soil Metagenome Grand Challenge[1]. It contains a big amount of data (456 GB) with a sufficient quality[2]; therefore it could be a suitable candidate for discovery of novel haloalkane dehalogenases.

However, since there is no prior knowledge about its contents, it would not be possible to use it to fully determine performance of the search system. To give an example, if a search would return zero results, it is not sure, whether this is caused by some error in the system or by real absence of the enzyme in the sample. In order to address this problem, I have decided to create artificial genome from sequences of seven known dehalogenases. In between these sequences, I have included a large amount of random nucleotides, so that its configuration will resemble the diverse character of a common metagenomic sample. Afterwards, I have sequenced the created file *in silico* using the sequencing simulation program *Grinder* implemented by Florent E. Angly et al. [6].

[1] http://genome.jgi.doe.gov/GrePraGChallenge_2/GrePraGChallenge_2.info.html

[2] Quality was assesed by FastQC [5] quality checking program.

Sample	Accession	Size	Description
Prarie metagenome	SAMN00737776	456 GB	Study assesing impact of land management on soil.
Lung metagenome	SAMN01923112	1.1 GB	Study to find the cause of death.
Generated sample	N/A	5.2 MB	Testing dataset consisting of dehalogenases with PDB accessions 3WI7, 4H7F, 4H77, 4MJ3, 5ESR, 5LKA and random nucleotides.

Table 7.1: Table summarizes information about all three testing samples. While the sample from Kansas has the biggest potential to contain novel dehalogenases, it is unsuitable for testing due to the absence of prior information about its contents. The second sample, metagenome from lungs of Terézia Hausmann, has bigger chance of containing a dehalogenase, because of the abudance of *mycobacterium tuberculosis*. The third sample was artificially constructed from sequences of known dehalogenases and random nucleotides and its sequencing was simulated *in silico* by sequencing simulator Grinder.

While the use of an artificial sample enables comparison of obtained results with a "ground truth", its character may still be different from samples found in nature. For this reason, I have decided to use a second real sample (NCBI accession SAMN01923112) that was extracted from mummified remains of 18th-century Hungarian woman Terézia Hausman and analyzed in study by Jacqueline Z.-M. et al. [10]. The reason, that makes it suitable for testing, is an unusually high concentration (8%) of genes belonging to *mycobacterium tuberculosis*. This bacteria is known to posses genes for dehalogenases; therefore this sample is very likely to contain a dehalogenase as well [24].

7.1.1 Results

Before the experimentation, samples were pre-processed on MetaCentrum grid using the pre-processing script `dispatcher.sh`. Quality thresholds needed for low-quality region removal were determined based on quality analysis conducted using the software FastQC [5]. The technical sequence removal was performed using Illumina TruSeq3 standard adapter sequences, which are shipped as part of the Trimmomatic software package.

Following the pre-processing, the soil sample from Kansas was searched using a multiple sequence alignment of 1,973 dehalogease sequences, which was created and kindly provided by experts from Loschmidt Laboratories [35]. The search resulted in one suitable candidate. In order to determine whether the found protein is a novel enzyme or an existing one, a search for similar sequences using the BLAST algorithm was conducted over the NCBI protein sequence database [40]. The closest match (70% identity) was a dehalogenase produced by bacteria *sphingobium japonicum* (PDB accession 1G42).

Next, the found sequence and the sequence from NCBI were aligned using the Needleman-Wunch global alignment algorithm, so that it would be possible to compare their catalytic sites. This revealed, that the found enzyme is missing important catalytic residues and probably would not perform its catalytic function. Details about missing residues are presented on the figure 7.1.

Nucleophile D108, after-nucleophile halide W109, before-cap acid E132, no base 272 and halide 38.

```
LinB    42   SYLWRNIMPHCAGLGRLIACDLIGMGDSDKLDPSGPERYAYAEHRDYLDALWEALDLGDR
             SYLWRNI+PH AGLGR +A DL+GMG+S +  P+G   Y +A+H  YLDA ++AL L +
Kansas  1    SYLWRNIIPHVAGLGRCLAPDLVGMGESGR-SPTG--SYRFADHSRYLDAWFDALGLTNN

LinB   102   VVLVVHDWGSALGFDWARRHRERVQGIAYMEAIAMPIEWADFPEQDRDLFQAFRSQAGEE
             VVLV+HDWGSALGF WA RH ERVQ IAYMEAI  P  W DFP     +F++ RS  GE
Kansas  58   VVLVLHDWGSALGFHWAYRHPERVQAIAYMEAIVQPRRWEDFPAGRDAMFRSLRSAQGER

LinB   162   LVLQDNVFVEQVLPGLILRPLSEAEMAAYREPFLAAGEARRPTLSWPRQIPIAGTPADVV
             LVL DN F+E VLP  I+R L++ EM AYR PF  + EAR PTL WPR++PI G PADVV
Kansas 118   LVLDDNFFIETVLPKSIIRTLTDDEMNAYRAPF-TSREARLPTLVWPRELPIDGEPADVV

LinB   222   AIARDYAGWLSESPIPKLFINAEPGALTTGRMRDFCRTWPNQTEI  266
             ++   Y  W+S++ IPKLFI AEPGA+  GR R+FCRTWPNQ E+
Kansas 177   SVVDAYGAWMSQTAIPKLFIAAEPGAILVGRAREFCRTWPNQREV  221
```

Figure 7.1: Alignment of the found sequence with a linB dehalogenase from bacteria *sphingobium japonicum* (PDB accession 1G42). The found sequence lacks two important residues that constitute catalytic pentad of dehalogenases – a halide near position 38 and a base near position 272. Because of this, the found enzyme would probably not have the desired catalytic function [46]. Present catalytic residues are typesetted in bold.

The same searching process was performed with the lung sample from mummified remains of Terézia Hausmann. The search resulted in one full enzymatic sequence of haloalkane dehalogenase. Further investigation using the NCBI BLAST has shown 100% similarity with HDL from *mycobacterium tuberculosis* (product of gene Rv2579, figure 7.2). Given the abudance of tuberculosis genome in the sample, this is an expected result. However, the fact that the gene from 200 year old remains has the same sequence as the one in contemporary m. tuberculosis bacteria may be interesting for further research.

```
>M. tuberculosis H37Rv|Rv2579|dhaA
MTAFGVEPYGQPKYLEIAGKRMAYIDEGKGDAIVFQHGNPTSSYLWRNIMPHLEGLGRLV
ACDLIGMGASDKLSPSGPDRYSYGEQRDFLFALWDALDLGDHVVLVLHDWGSALGFDWAN
QHRDRVQGIAFMEAIVTPMTWADWPPAVRGVFQGFRSPQGEPMALEHNIFVERVLPGAIL
RQLSDEEMNHYRRPFVNGGEDRRPTLSWPRNLPIDGEPAEVVALVNEYRSWLEETDMPKL
FINAEPGAIITGRIRDYVRSWPNQTEITVPGVHFVQEDSPEEIGAAIAQFVRRLRSAAGV
```

Figure 7.2: Sequence of dehalogenase, which is the product of gene Rv2579 from *mycobacterium tuberculosis* reference genome H37Rv. Search using the lung sample resulted in one match identical to this sequence. No novel variants were found.

Finally, the artificial sample was used to determine the accuracy of the search system. Because the sample was generated, its contents are known and if the system works correctly, the set of resulting candidate sequences must contain only those, which were used to generate the sample. However, results have shown, that this is not the case. Depending on settings of assembly parameters, approximately 30–50% of sequences were chimeric. This suggest, that the found dehalogenase from the Kansas sample may also by chimeric and not

an enzyme commonly found in nature. Interestingly, this problem does not apply to the result from lung metagenome, because it was identical to an existing enzyme. The possible explanation to why the assembly have not failed in this case could be, that there were no other variants of dehalogenase in the sample, and so it was not possible to assemble parts from different genes into a chimeric sequence. Overall, the problem of chimeric sequences may be caused by inability of assembly program Velvet to work with metagenomic data, as it was developed to function with genomes.

For this reason, I have decided to experiment with an alternative algorithm for gene extraction from metagenomic samples, called *Gretel*, published by Nicholls, et al. in 2016 [42]. In contrast to assembly, which uses found reads directly, Gretel requires them to be firstly aligned to the input search query sequence. If the user conducted the search using a multiple sequence alignment, it is possible to proceed by choosing one sequence from the alignment, or its consensus. Next, sequences of aligned reads are compared with corresponding parts of the input query. Output of this comparison is a set of bases from reads, which differ from corresponding bases on the query sequence. These differences are commonly called *Single-Nucleotide Polymorphisms*, or SNPs. The Gretel algorithm uses reads together with their SNPs to create a graph data structure. In turn, this structure is traversed, and the information stored in it is used to assemble reads into longer sequences. Assembly is performed in a way, which maximizes likelihood of resulting contigs being genuine non-chimeric sequences.

While experimenting with Gretel on the Kansas prairie sample provided no search results, experimentation on the lung sample resulted in two candidates identical to Rv2579 tuberculosis gene. After a closer inspection, it became apparent, that in the case of the Kansas sample, the used alignment program Bowtie 2 [28] was not able to align any reads. Like in the case of Velvet, the reason for this may lie in fact, that it was implemented for standard genomic data. In essence, its alignment strategies may be too conservative to align diverse variants of genes that could be found in a metagenomic sample.

Interestingly, the use of the algorithm on the generated dataset have led to no results despite successful alignment. In order to find a possible reason, I have compared the alignment of reads in the case of lung and generated sample. While the generated sample contained minimal number of SNPs, the lung metagenome contained uniform distribution of SNPs across reads. These were mostly positioned on the third nucleotide in codon. Since, codons encoding the same amino acid usually differ only in the third nucleotide, most of these mutations were probably neutral (i.e. not causing change of amino acid) [3]. Even though they were neutral, their presence may have helped Gretel to build the graph structure. In case of the generated sample, the small number of SNPs may have rendered the construction impossible.

7.2 Catalytic Function Verification

Proper assesment of verification methods requires a suitable dataset of enzyme sequences, which would provide catalytic region annotations and would guarantee sufficient quality of data. Based on these requirements, I have decided to use the data from Swiss-Prot protein sequence database, which has its entries manually reviewed by human experts [44]. Enzymes in Swiss-Prot are divided into groups based on chemical reactions they catalyse. Each group is identified by a sequence of four numbers called the Enzyme Commision (EC) number [54]. In the written form, these numbers are separated by the dot character and preceded by the string “EC”. For example, the identifier corresponding to Haloalkane Dehalogenases is

"EC 3.8.1.5". This method of classification was developed by the International Union of Biochemistry in 1965 and its use is widespread [43].

For the purpose of evaluation, all enzymes with their annotations were downloaded from the Swiss-Prot database and grouped by their EC number. The resulting set contained 5038 groups and 216 208 records[3]. In order to reduce the size of the dataset and increase the variability of sequences within groups, each group was clustered using the protein clustering program CD-HIT [32]. Clustering was performed based on 90% sequence similarity. This means, that sequences which had 90% or more identical (or closely-related) residues, were assigned to the same cluster. After clustering, all, but representative sequences of clusters, were removed from the dataset. This resulted in approximately half of sequences being discarded.

Furthermore, I have decided to limit the dataset to enzymes, which have only one catalytic region. Not only this significantly reduces its size, but also helps to hihlight differences in performance among classifiers by providing the biggest classification challenge. This is because all of the implemented verification methods take into account all catalytic regions of the input enzyme, and therefore, the greater the number of catalytic regions there is, the more accurate the classification will be. For instance, if there are three catalytic regions on an enzyme and one is misclassified, the correct classification of remaining ones can still prevent the incorrect classification of the whole enzyme. In this way, if a number of regions is sufficient, even poorly-performing classifier can seem to have similar performance as the one, which is very accurate. The final dataset was stored in a special JSON format, and is avaliable in the file `data/active_clustered.json`.

After the final dataset was prepared, it was used in a number of classification experiments. Each experiment followed the same evaluation protocol depicted on the figure 7.3. In its first step, a random subset of the dataset was extracted for evaluation. This is neccessary, because even though the amount of data have decreased during the preparation, it was still too large to be evaluated in reasonable time. In all cases, the resulting subset consisted of 10 randomly selected EC groups, with each containing 30 randomly selected enzymes. Next, the obtained subset was split into two subsubsets. While, each of these contained the same EC groups, enzymes within groups were split in the ratio 1:2 (10 enzymes in the first, 20 in the second). I will refer to the first, smaller, subsubset as the set of *candidate enzymes* and to the second as the set of *known enzymes*. In the following step, catalytic function of each enzyme from the set of candidate enzymes was verified by selected verification method using all of the enzymes from the set of known enzymes. Verification using an enzyme from the same EC group was expected to have positive class label, while verification with enzymes from outside of candidate's group the negative one. In order to counter possible sampling bias, the process of random subset extraction, splitting and evaluation was repeated 10 times in each experiment. Finally, the obtained experimental results were used to calculate the *Reciever operating characteristic* (ROC) and *Area under the curve* (AUC) performance metrics. These were averaged across repetitions of each experiment.

[3] Records were downloaded in December 2016.

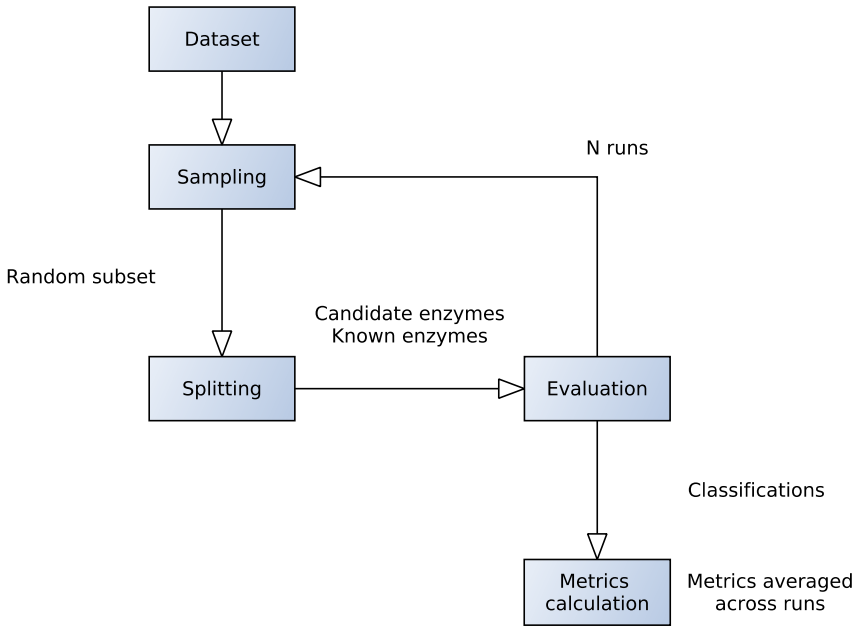

Figure 7.3: Diagram of the evaluation protocol. In the first step, a random subset is selected from the testing dataset. Then, it is split into the set of candidate and known enzymes. Next, in the evaluation step, all enzymes from the set of candidate enzymes are classified using the set of known enzymes. In order to avoid possible sampling bias, in each experiment, the whole process is repeated for a number of times. In the final step, obtained data is used to calculate performance metrics of classifiers.

7.2.1 Methods Based on Global Alignment

The first experiment was conducted using the verification method based on the global alignment and its main goal was to determine the effect of different scoring matrices on the performance of the classifier. Used matrices belonged to two of the most common scoring matrix families – Point Accepted Mutation (PAM) and Blocks Substitution Matrix (BLOSUM). While PAM matrices were created with regards to evolutionary distance, the BLOSUM family was built solely based on alignment of conserved protein sequences [21]. In both cases, individual matrices are further identified by a number. While, in the case of PAM, this number denotes the evolutionary distance, in case of BLOSUM, the number expresses the minimal similarity of sequences, which were used to build the matrix. For my experiment, I have chosen three matrices from the PAM family – matrix for short evolutionary distance (PAM 30), medium evolutionary distance (PAM 120) and long evolutionary distance (PAM 250) – and analogically, three from the BLOSUM family – a matrix for closely releated proteins (BLOSUM 100), related (BLOSUM 62) and distantly related proteins (BLOSUM 30).

First, the performance of variants using all of these matrices, together with the variant using the simple scoring model, was evaluated for different lengths of the classification window. In all cases, the fixed windowing strategy was used. Evaluation results are presented on the graph 7.4, which shows their performance expressed by the metric AUC. For all window sizes, the matrix BLOSUM 62 has achieved the best classification results. This is probably because the matrix BLOSUM 62 provides the most accurate alignment, which

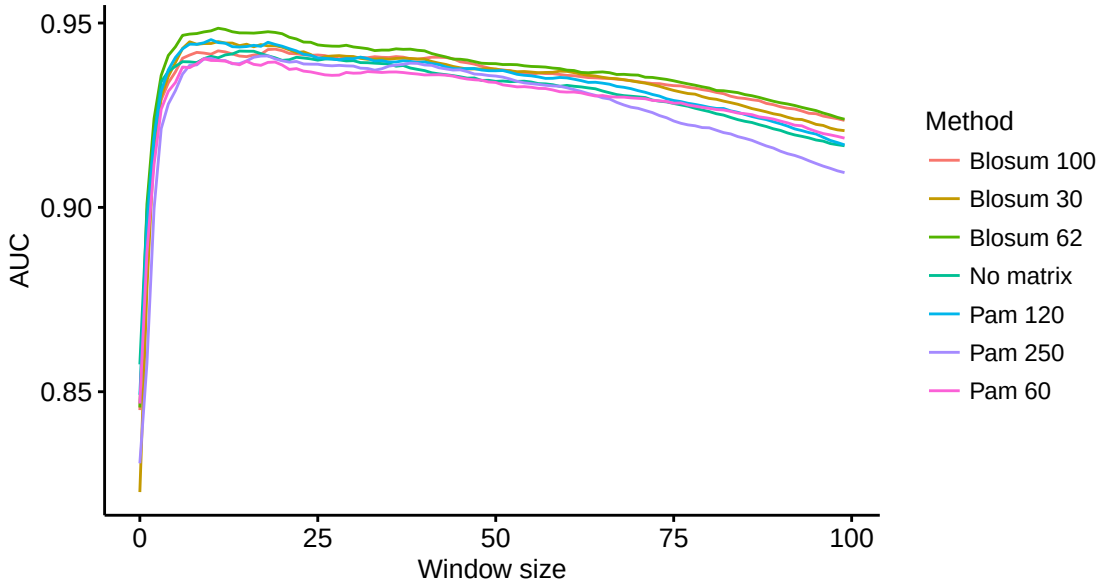

Figure 7.4: The graph shows the average performance of unweighted global alignment methods for different scoring matrices, expressed as the AUC metric averaged across 10 runs of the experiment. Results are presented with respect to the length of the classification window. The best performance was achieved with the matrix BLOSUM 62, regardless of the window size.

is cruicial for scoring, as catalytic residues can be correctly compared only when they are correctly aligned. This claim is further supported by the study presented by S. Henikoff and J. G. Henikoff [22], in which they compared the performance of PAM and BLOSUM matrices and came to conclusion that BLOSUM 62 is the best performing matrix for both homology search and alignment. The maximal AUC achieved was 0.94 with window size 11.

Next, the same experiment was repeated using the soft windowing strategy. In this case, the best achieved AUC among all matrix variants was 0.6, which is only slightly better than an AUC of a classifier based on random guessing (AUC 0.5). The reason for this poor performance probably lies in the fact, that soft window methods take into account only positions, where sequences of both, the known enzyme and the enzyme candidate, have no gaps. In effect, the information about insertions and deletions is discarded, even though it might be important for the classification.

Finally, in the last experiment, the best global alignment method with matrix BLOSUM 62 was evaluated for window size values in range 0 to 30 using different scoring weights of the catalytic region. The experiment was also repeated for the weighted variant with multiplication by the partial score of the catalytic region. Maximal AUC obtained by each configuration is presented in the table 7.2. The best results were achieved with no multiplication and the weight 2. The graph 7.5 compares the performance of this variant with its unweighted counterpart. For all sizes of the classification window, the weighted variant shows better performance. Its maximal AUC of 0.95 was achieved with the window size 11, and constitutes an incerease of 0.01 compared with the maximal value of the unweighted variant.

While the use of weight 2 led to best classification results, weights 4 and 5 manifested worse performance than unweighted scoring. Furhermore, in all cases, the variant with

	1	2	3	4	5
With multiplication	0.930	0.926	0.921	0.917	0.914
Without multiplication	0.949	**0.950**	0.950	0.949	0.948

Table 7.2: The table summarizes results of the experiment with weighted global method (matrix BLOSUM 62) for weights in the range 1–5 and for both, the variant with multiplication by the partial score of the catalytic region and the variant without it. Experiment was repeated 10 times for each weight setting using all window sizes in the range 0–30. Values in the table represent the maximal average AUC that was encountered for the given weight setting. The best result was achieved with weight 2 and no multiplication (AUC 0.9499). However, the variant with the weight 3 reached nearly the same score (AUC 0.9498).

multiplication, which strongly emphasizes the score of the catalytic region, have achieved worse AUC then the variant without it. Both of these facts suggest, that while higher weight of the catalytic region can improve classification results, exaggerated emphasis on it may lead to the opposite.

7.2.2 Method Based on Local Alignment

The evaluation of methods based on the local alignment was conducted using the same matrices as the evaluation of global methods. While, in the case of global methods, the presence of scoring matrix helped to achieve better results, in this case it proved to act in the opposite manner. This effect is illustrated on the graph 7.6. As it shows, the local method with no matrix performed significnatly better than any of the variants with matrix. Its maximum average AUC value was 0.93 (window size 9), while the best matrix-based variant (BLOSUM 62) have reached only value 0.77 (window size 7).

The possible explanation of this result lies in the fact, that catalytic regions tend to be very conserved. This means, that two enzymes with the same catalytic function will probably have a number of identical residues around the position of their catalytic regions. Therefore, when trying to align the catalytic region of one enzyme with the whole sequence of the other, it might be most effective to do it strictly based on symbol comparison and not to take into account other properties of residues. While the simple scoring model works exactly in this manner, the scoring matrices use different scoring values for matches and mismatches based on residues involved. For instance, the matrix BLOSUM 62 assigns score value 11 for a match between two tryptophan residues, but only 4 for two alanine residues. Likewise, mismatch between lysine and arginine has a positive score of 2, while mismatch between serine and tryptophan has negative penalty of -3. In this way, scoring matrices allow to align proteins which are homologous, but do not have identical sequences. However, the ability to match sequences which do not share high identity on the level of symbols may, in the case of short conserved catalytic regions, lead to false alignments.

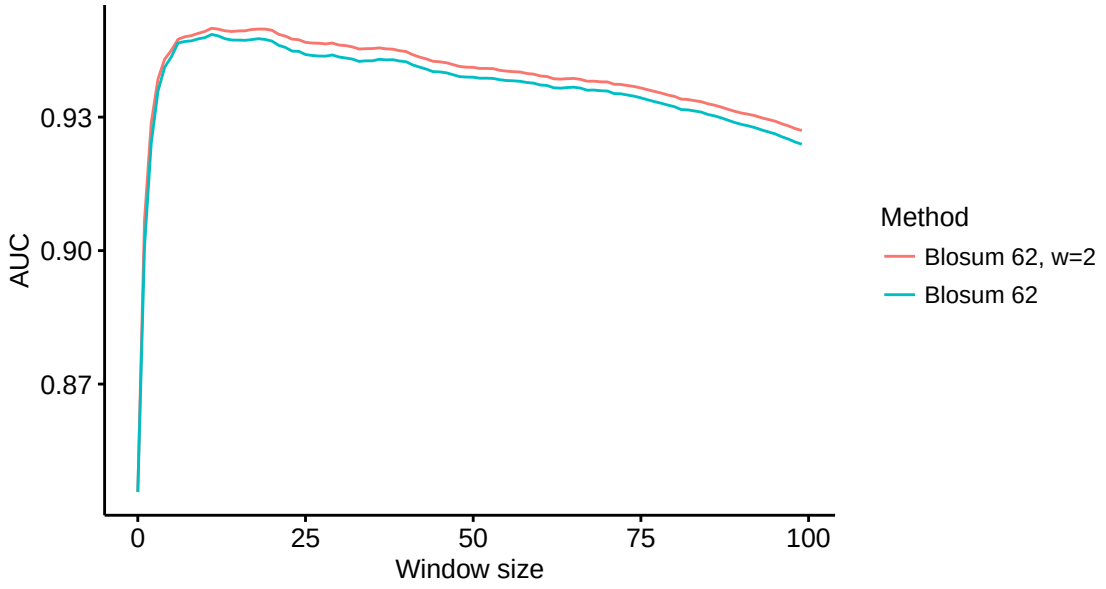

Figure 7.5: The graph compares performance of the best unweighted global method (matrix BLOSUM 62) with performance of its best weighted variant (matrix BLOSUM 62; weight 2; no multiplication) for different lengths of the classification window. The metric used is AUC averaged across 10 runs of the experiment. The weighted method achieves higher scores for all sizes of the classification window. Its maximal score is 0.95, which represents an increase of 0.01 compared with the unweighted variant.

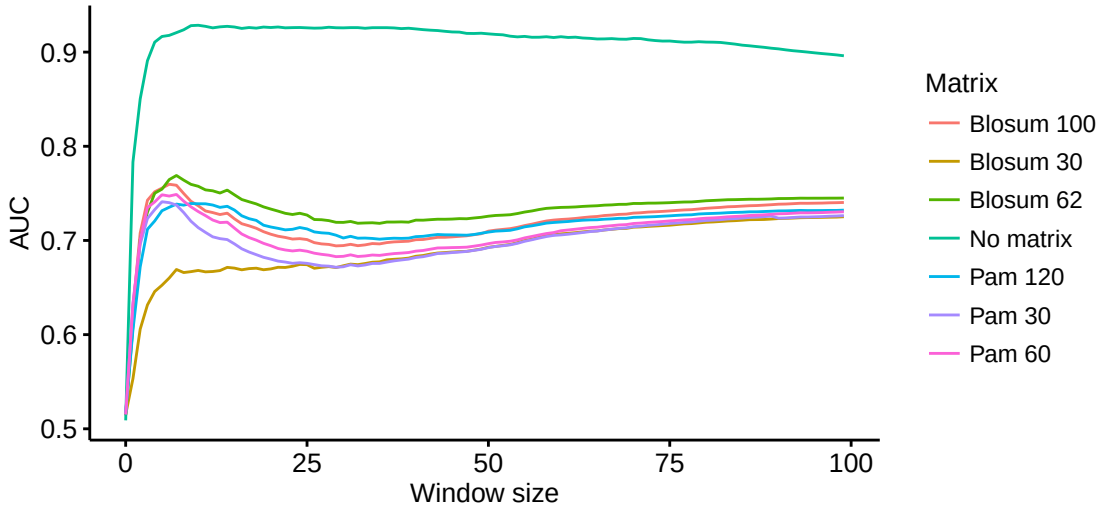

Figure 7.6: The graph shows the average performance of local alignment methods for different scoring matrices, expressed as the AUC metric averaged across 10 runs of the experiment. Results are presented with respect to the length of the classification window. For all window lengths, the best performance was achieved without matrix. Maximal AUC value of this variant was 0.93.

7.2.3 Method Based on Normalized Cross-Correlation

The first step of the correlation method, described in the section 5.3.2, is translation of sequences into a matrix form, in which, each residue is represented as one column vector. Values of this vector correspond to some measured physical or chemical properties of the given residue. This form of representation, as well as, the first set of vectors was introduced in the paper by Kidera et al. [25]. In my experiment, I have used their original values, as well as a newer set, which is part of the PredictSNP disease-releated mutation classifier implemented by Bendl et al. [8].

Graph 7.7 compares the performance of these two variants with respect to the size of the classification window. For all window sizes, the original set of vectors showed significantly better performance. Its maximal average AUC value was 0.94 (window size 9), while for the set from PredictSNP it was only 0.81 (window size 65). Deeper analysis of reasons why the original set of vectors greatly outpreformed the set from PredictSNP would require expert knowledge from biology and chemistry, and is beyond the scope of this thesis.

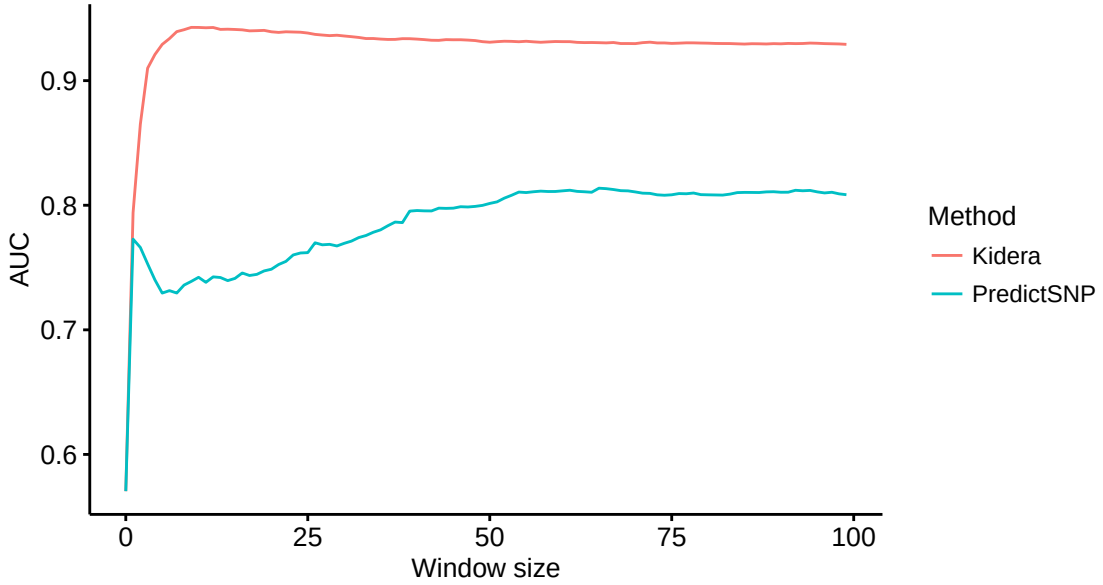

Figure 7.7: The graph compares the average performance of the correlation method with residue vectors introduced by Kidera et al. [25], and its variant with vectors from the PredictSNP [8] disease-releated mutation classifier. The metric used is AUC averaged across 10 runs of the experiment. Performance is presented as a function of the classification window length. The set of vectors presented by Kidera et al. had better performance for all lengths of the classification window. Its best achieved AUC was 0.94 for window length 9. The best AUC of the second method was 0.81.

7.2.4 Comparison of Verification Methods

The goal of the final analysis was to take a closer look at the best performing variants from each of the classification methods, in order to compare their properties and create a set of recommendations for their use. The analyzed variants were: global verification method using the BLOSUM 62 matrix with catalytic region weight 2 and no multiplication, local verification method with no matrix and correlation method with the original set of vectors

Method	Matrix	Window	AUC
Global	Blosum 62	11	0.950
Correlation	N/A	9	0.943
Local	No matrix	9	0.928

Table 7.3: The table presents maximal encountered average AUC values of the three best verification methods – the global method with matrix BLOSUM 62, weight 2 and no multiplication; the correlation method with vectors introduced by Kidera et al. [25] and the local method with no scoring matrix. The column "Window" specifies the classification window length, for which the result was achieved. Rows are sorted in descending order by their AUC value.

proposed by Kidera et al. For the sake of clarity, in the rest of this section, I will refer to them only as the global, local and the correlation method, without repeating their detailed specifications.

Firstly, the performance of these methods was compared using the AUC metric, in order to find out which one of them is the most accurate. While, for each method, the AUC metric varies depending on the size of the classification window, only its maximal value is relevant for this comparison. Based on the maximal average AUC, the best result was achieved by the global method, second best by the correlation method and the worst by the local method. Maximal AUC values, together with corresponding window lengths are presented in the table 7.3. Overall characterization of methods with regard to the classification window length is presented in the graph 7.11.

Closer analysis of these results have shown, that the difference between maximal AUC of the global method and correlation (0.007) is more than two times smaller than the difference between correlation and the local method (0.02). The significance of these differences is more apparent from the trade off between false and true positive rates for each of these methods. To give a concrete example, if the classification threshold would be set to a value that would guarantee false positive rate of no more than 5%, the global method would be expected to classify 89% of enzymes with the same function correctly, the correlation method 88% and the local method 80%. While the difference of 1% between the global and correlation method might be, from the practical standpoint, insignificant, the same cannot be easily said about the difference of 8% between the global and correlation method. ROC curves expressing general trade off between false positives and true positives for all methods are depicted on the graph 7.8.

Since all of these variants achieve AUC higher than 0.9[4], it is possible to use them in a real-world application. However, in order for any program to be truly usable, it must provide a reasonable speed of execution. As further analysis have shown, this property varies greatly among evaluated methods. For instance, the most accurate classification results of the global method come with the cost of the lowest throughput of 9.95 verifications per second on the average. In contrast, the correlation method, which proved to be the fastest, is able to perform 1,203.05 verifications, which is approximately 120 times more. The local scoring method has an average throughput of 121.37 which is approximately 10 times less then the correlation method and 12 times more than the global method. More details about

[4] An ideal classifier would have AUC value of 1.

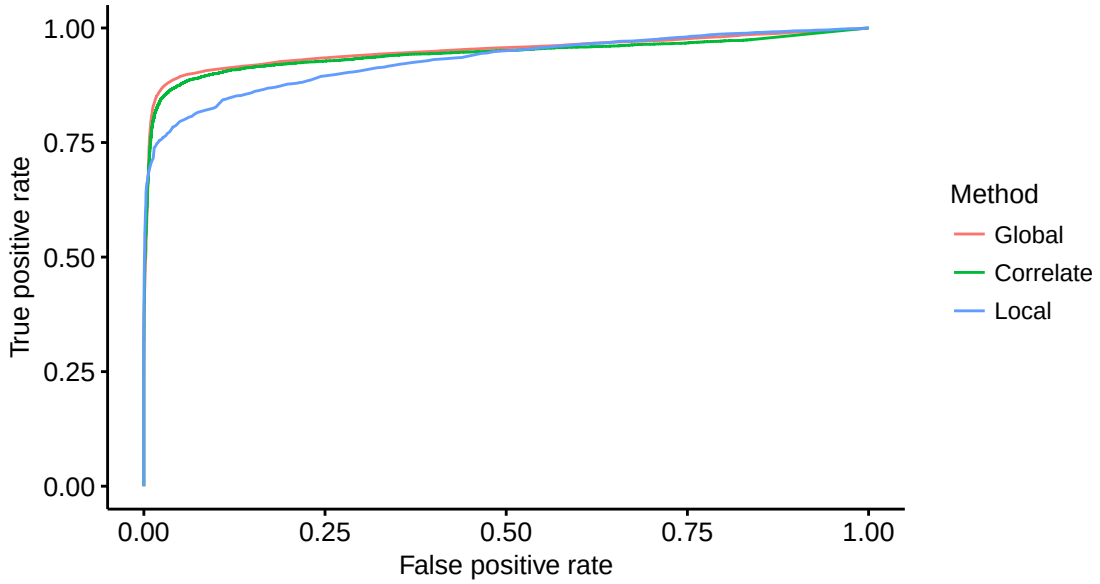

Figure 7.8: The graph of ROC curves for the three best verification methods – the global method with matrix BLOSUM 62, weight 2, window 11 and no multiplication; the correlation method with vectors introduced by Kidera et al. and window 9 [25] and the local method with no scoring matrix and window 9. While the difference between the global and correlation method is small and might be insignificant for many practical purposes, the difference between the local method and other methods can not be easily overseen.

the throughput of methods are presented on the graph 7.9. The analysis was conducted using a regular personal computer (Intel Core I5, 8 GB RAM, 64-bit Linux).

Furthermore, I have decided to analyze the effect of window length on the execution time of each method[5]. While both, the local and the correlation method, show linear growth of required execution time as a function of the window size, the execution time of the global method appears to be constant (graph 7.10). The reason for this effect is the fact, that the global scoring method uses the Needleman-Wunch alignment algorithm to align enzyme sequences over their whole length regardless of the classification window size. The subsequent windowing and scoring operations use only comparison and basic arithmetic operations, which are, compared to the global alignment algorithm, very fast, and so their contribution to the overall complexity is insignificant. In contrast, the local scoring method aligns only the window area of the known enzyme to the whole sequence of the enzyme candidate, and therefore the size of the window directly affects the execution time of the complex alignment operation. The situation of the correlation method is analogical, because it is built on the same basic principle, but uses correlation instead of alignment.

In conclusion, my experiments have shown that the weighted global scoring method with matrix BLOSUM 62 achieved the best classification performance. However, the novel correlation approach had only slightly lower accuracy and provided approximately 120 times faster execution speed. Based on these properties, I conclude, that the correlation method might be the most suitable approach for catalytic function verification in most practical applications and the global method should be reserved for uses, where execution time is

[5]The execution time of a verification method is the multiplicative inverse of its throughput.

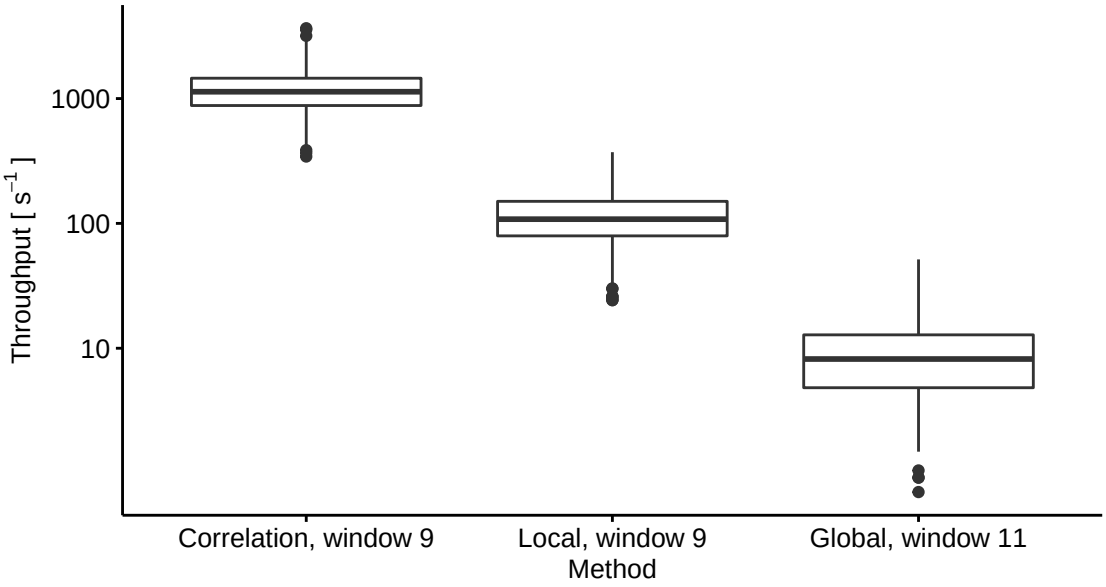

Figure 7.9: Boxplot depicts the throughput (number of classifications per second) of each of the three best verification methods – the global method with matrix BLOSUM 62, weight 2 and no multiplication; the correlation method with vectors introduced by Kidera et al. [25] and the local method with no scoring matrix. As it illustrates, the highest value was achieved by the correlation method ($\mu = 1,203.05$, $\sigma = 463.34$), the second highest by the local method ($\mu = 121.37$, $\sigma = 60.39$) and the lowest by the global method ($\mu = 9.95$, $\sigma = 7.29$). Throughput of each method was measured on 1,000 classifications. Enzymes for the experiment were randomly chosen from the SwissProt database [44]. The analysis was conducted using a regular personal computer (Intel Core I5, 8 GB RAM, 64-bit Linux). Please note that the graph uses logarithmic scale with base 10.

not critical, and even small increase of accuracy is appreciated. The local method proved to have the lowest performance, and therefore the other two should be given priority.

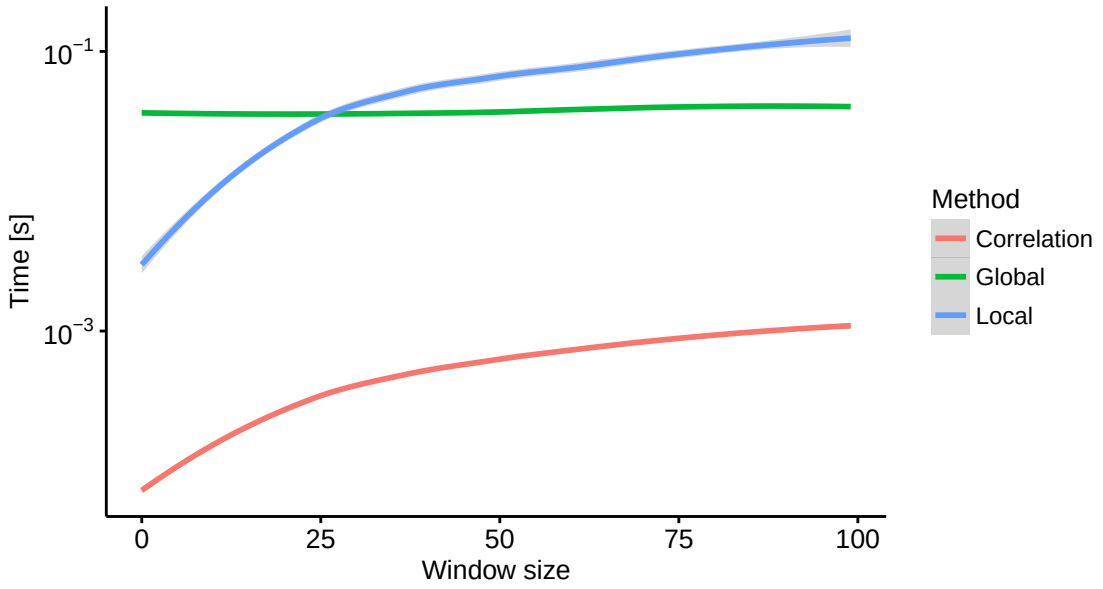

Figure 7.10: The graph presents mean execution time of a single verification as the function of the classification window length for all three best classification methods – the global method with matrix BLOSUM 62, weight 2 and no multiplication; the correlation method with vectors introduced by Kidera et al. [25] and the local method with no scoring matrix. For all lengths, the correlation method has the fastest execution. While the execution time of both, the correlation method and the local method, linearly increases with increasing size of the classification window, the execution of the global method seems to have a constant character. The analysis was conducted using a regular personal computer (Intel Core I5, 8 GB RAM, 64-bit Linux). Please note that the graph uses logarithmic scale with base 10.

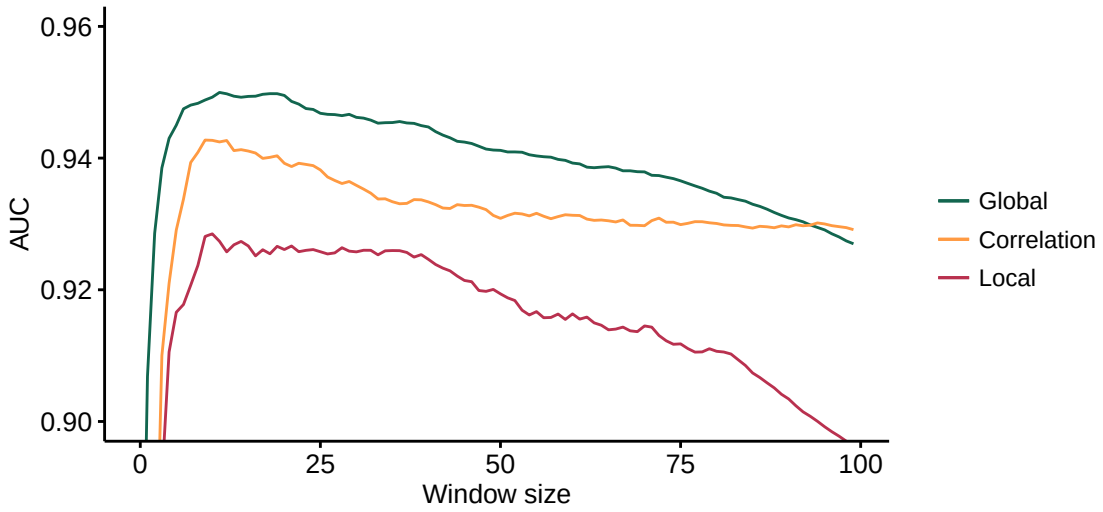

Figure 7.11: The graph shows the average performance of the best methods – the global method with matrix BLOSUM 62, weight 2 and no multiplication; the correlation method with vectors introduced by Kidera et al. [25] and the local method with no scoring matrix. The metric used was AUC averaged across 10 runs of the experiment. Results are presented with regards to the length of the classification window. Overall, the best performance was achieved by the global method, the second best by the correlation and worst by the local method. While the difference of the maximal AUC between correlation and global method was only 0.007, in case of the correlation method and local method it was 0.02.

Chapter 8

Conclusion

The main goal of this thesis was to design and implement a set of tools for enzyme detection in metagenomic data. In order to do so, it was first neccessary to gather extensive knowledge from fields of molecular biology and bioinformatics. Results of this effort are presented in chapters 2, 3 and 4. The gathered information was than utilized to create a specification of the system, which is presented in chapter 5. The specification have identified three key elements that need to be implemented – read pre-processing, homology search and enzymatic function verification.

The task of the pre-processing module (section 5.1), is to take raw reads from metagenomic sample, pre-processes them with regards to their sequencing quality and output them in a searchable format. Pre-processing is cruicial, because presence of low-quality reads could undermine reliability of homology search results [38].

The goal of the homology search module is to use the pre-processed sample to find novel enzymes based on a sequence of some known enzyme provided by the user. Instead of a single sequence, the specified system also allows to use a multiple sequence alignment. The result of its search consists of candidate enzyme sequences, which are similar to the input sequence, or would fit into the input multiple sequence alignment.

However, sequential similarity to the input enzyme does not guarantee that found candidates also perform the same catalytic function. This problem is addressed by the enzymatic function verification module. Based on their catalytic residues, the verification module determines which of found enzyme candidates truly perform the desired catalytic function. For this purpose, I have proposed two classes of catalytic function verification methods – verification based on alignment and verification based on the normalized cross-correlation. Their specification, implementation and evalution is one of the main contributions of my master's thesis.

All three proposed modules were successfully implemented in the Python 3 programming language with some parts in BASH and C. The basic functionality of the implementation was tested on a personal computer, as well as, on the MetaCentrum grid computing infrastructure. Afterwards, a more in-depth evaluation of the homology search and enzymatic function verification module was conducted in order to analyze their limits and abilities. The evaluation of pre-processing module was omitted, because methods used in it are standard, their use is widespread and they have been already evaluated [9].

The first experiment was conducted using the homology search module and led to discovery of a novel haloalkane dehalogenase enzyme in metagenomic soil sample from prairie in Kansas. Further investigation revealed that the found sequence is similar to an enzyme expressed by *sphingobium japonicum* bacteria. However, it soon became apparent, that the

found sequence is incomplete and is missing some of important catalytic residues. Because of this, it would probably not be able to perform its catalytic function. Moreover, further evaluation of the module using other samples have shown that it is prone to outputting chimeric sequences. These limitations are probably caused by used methods, which originally come from genomics and might not be suitable for use in metagenomics. Even though found sequences might be in many cases chimeric, they, and especially their catalytic site variants, can still serve as valuable information for protein engineering.

The second set of experiments was aimed at enzymatic function verification methods and have proven, that most of them achieve very good accuracy and are usable in real-world applications. While the best of these methods was based on alignment, the novel correlation approach had only slightly lesser accuracy and provided approximately 120 times faster speed of execution. Moreover, its function is not limited to complete sequences, and therefore, it is able to identify catalytic sites even in protein fragments. For these reasons, the correlation method is very suitable for practical catalytic function verification, especially in settings with big amounts of data.

In the future extensions of the system, it would be most beneficial to address limitations of the homology search module. One of the possible ways of achieving this, could be modification of the pre-processing module to include full *de novo* metagenomic assembly. While this would probably increase the time required for pre-processing, the resulting set of contigs may have higher chance to be non-chimeric. This is because metagenomic assembly programs were specifically developed to work with metagenomic data and might be better suited to handle its specifics. However, further study and experimentation is required in order to determine to which extent this would be true.

For the enzymatic function verification, the future goal could be creation of a classifier, that would be able to directly determine catalytic function of a given sequence, without the need for user to provide annotated sequence of some known enzyme. For instance, this could be achieved by manually selecting fixed number of enzymes from each currently known enzyme group, and using them to create k-nearest neighbors classifier with score of catalytic function verification as the distance metric. Such classifier, together with the ability of the correlation method to work with protein fragments, could be used to classify all reads of metagenomic sample. The result of this would be a description of enzymatic composition of the sample, which might be a useful descriptor in areas such as medical diagnostics or agricultural soil analysis.

Bibliography

[1] A One-Letter Notation for Amino Acid Sequences. *European Journal of Biochemistry*. vol. 5, no. 2. 1968: pp. 151–153. ISSN 1432-1033. doi:10.1111/j.1432-1033.1968.tb00350.x.

[2] Alberts, B.; Bray, D.; Hopkin, K.; et al.: *Essential Cell Biology*. Garland Science. 2009. ISBN 0815341296.

[3] Alberts, B.; Johnson, A.; Lewis, J.; et al.: *Molecular Biology of the Cell*. Garland Science. 2014. ISBN 0815344325.

[4] Altschul, S. F.; Gish, W.; Miller, W.; et al.: Basic local alignment search tool. *Journal of molecular biology*. vol. 215, no. 3. 1990: pp. 403–410.

[5] Andrews, S.; et al.: FastQC: a quality control tool for high throughput sequence data [software]. Version 0.11.5.
Retrieved from: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[6] Angly, F. E.; Willner, D.; Rohwer, F.; et al.: Grinder: a versatile amplicon and shotgun sequence simulator. *Nucleic Acids Research*. vol. 40, no. 12. 2012: page e94. doi:10.1093/nar/gks251.

[7] Behnel, S.; Bradshaw, R.; Citro, C.; et al.: Cython: The Best of Both Worlds. *Computing in Science Engineering*. vol. 13, no. 2. March 2011: pp. 31–39. ISSN 1521-9615. doi:10.1109/MCSE.2010.118.

[8] Bendl, J.; Stourac, J.; Salanda, O.; et al.: PredictSNP: robust and accurate consensus classifier for prediction of disease-related mutations. *PLoS computational biology*. vol. 10, no. 1. 2014.

[9] Bolger, A. M.; Lohse, M.; Usadel, B.: Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*. 2014. doi:10.1093/bioinformatics/btu170.

[10] Chan, J. Z.-M.; Sergeant, M. J.; Lee, O. Y.-C.; et al.: Metagenomic Analysis of Tuberculosis in a Mummy. *New England Journal of Medicine*. vol. 369, no. 3. jul 2013: pp. 289–290. doi:10.1056/nejmc1302295.

[11] Chenna, R.; Sugawara, H.; Koike, T.; et al.: Multiple sequence alignment with the Clustal series of programs. *Nucleic acids research*. vol. 31, no. 13. 2003: pp. 3497–3500.

[12] Cock, P. J.; Antao, T.; Chang, J. T.; et al.: Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*. vol. 25, no. 11. 2009: pp. 1422–1423.

[13] Cock, P. J.; Fields, C. J.; Goto, N.; et al.: The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*. vol. 38, no. 6. 2010: pp. 1767–1771.

[14] Del Fabbro, C.; Scalabrin, S.; Morgante, M.; et al.: An extensive evaluation of read trimming effects on Illumina NGS data analysis. *PLoS One*. vol. 8, no. 12. 2013: page e85024.

[15] DeLano, W. L.: The PyMOL molecular graphics system [software]. Version 1.8.6. Retrieved from: https://pymolwiki.org/index.php/

[16] Eddy, S. R.: HMMER: Profile hidden Markov models for biological sequence analysis [software]. Version 3.1b2.
Retrieved from: https://www.ebi.ac.uk/Tools/hmmer/

[17] Frazzetto, G.: White biotechnology. *EMBO reports*. vol. 4, no. 9. 2003: pp. 835–837.

[18] Gao, J.; Li, Z.: Uncover the conserved property underlying sequence-distant and structure-similar proteins. *Biopolymers*. vol. 93, no. 4. 2010: pp. 340–347.

[19] Handelsman, J.; Rondon, M. R.; Brady, S. F.; et al.: Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & biology*. vol. 5, no. 10. 1998: pp. R245–R249.

[20] He, Y.; Rackovsky, S.; Yin, Y.; et al.: Alternative approach to protein structure prediction based on sequential similarity of physical properties. *Proceedings of the National Academy of Sciences*. vol. 112, no. 16. 2015: pp. 5029–5032.

[21] Henikoff, S.; Henikoff, J. G.: Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*. vol. 89, no. 22. 1992: pp. 10915–10919.

[22] Henikoff, S.; Henikoff, J. G.: Performance evaluation of amino acid substitution matrices. *Proteins: Structure, Function, and Bioinformatics*. vol. 17, no. 1. 1993: pp. 49–61.

[23] Hernandez, D.; François, P.; Farinelli, L.; et al.: De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome research*. vol. 18, no. 5. 2008: pp. 802–809.

[24] Jesenska, A.; Sedlacek, I.; Damborsky, J.: Dehalogenation of Haloalkanes by Mycobacterium tuberculosis H37Rv and Other Mycobacteria. *Applied and environmental microbiology*. vol. 66, no. 1. 2000: pp. 219–222.

[25] Kidera, A.; Konishi, Y.; Oka, M.; et al.: Statistical analysis of the physical properties of the 20 naturally occurring amino acids. *Journal of Protein Chemistry*. vol. 4, no. 1. 1985: pp. 23–55.

[26] Krane, D. E.; Raymer, M. L.: *Fundamental Concepts of Bioinformatics*. Pearson. 2002. ISBN 0805346333.

[27] Krogh, A.: An introduction to hidden Markov models for biological sequences. *New Comprehensive Biochemistry*. vol. 32. 1998: pp. 45–63.

[28] Langmead, B.; Salzberg, S. L.: Fast gapped-read alignment with Bowtie 2. *Nature methods*. vol. 9, no. 4. 2012: pp. 357–359.

[29] Leinonen, R.; Sugawara, H.; Shumway, M.: The Sequence Read Archive. *Nucleic Acids Research*. vol. 39, no. suppl_1. 2011: page D19. doi:10.1093/nar/gkq1019.

[30] Lewis, J.: Fast normalized cross-correlation. In *Vision interface*, vol. 10. 1995. pp. 120–123.

[31] Li, H.; Homer, N.: A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*. vol. 11, no. 5. 2010: pp. 473–483.

[32] Li, W.; Godzik, A.: Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*. vol. 22, no. 13. 2006: pp. 1658–1659.

[33] Li, Z.; Chen, Y.; Mu, D.; et al.: Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*. vol. 11, no. 1. 2012: pp. 25–37.

[34] Lorenz, P.; Eck, J.: Metagenomics and industrial applications. *Nature Reviews Microbiology*. vol. 3, no. 6. 2005: pp. 510–516.

[35] Loschmidt Laboratories: Protein Engineering Group [online]. Acessed on 2017-05-19. Retrieved from: https://loschmidt.chemi.muni.cz/peg/

[36] Madden, T.: The BLAST sequence analysis tool [software]. Version 2.2.31+. Retrieved from: https://blast.ncbi.nlm.nih.gov/

[37] McCarthy, B.; Holland, J.: Denatured DNA as a direct template for in vitro protein synthesis. *Proceedings of the National Academy of Sciences*. vol. 54, no. 3. 1965: pp. 880–886.

[38] Metzker, M. L.: Sequencing technologies—the next generation. *Nature reviews genetics*. vol. 11, no. 1. 2010: pp. 31–46.

[39] Nagarajan, N.; Pop, M.: Sequence assembly demystified. *Nature Reviews Genetics*. vol. 14, no. 3. 2013: pp. 157–167.

[40] National Center for Biotechnology Information: Protein database [online database]. Acessed on 2017-04-12.
Retrieved from: https://www.ncbi.nlm.nih.gov/protein/

[41] Needleman, S. B.; Wunsch, C. D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*. vol. 48, no. 3. 1970: pp. 443–453.

[42] Nicholls, S. M.; Aubrey, W.; de Grave, K.; et al.: Advances in the recovery of haplotypes from the metagenome. *bioRxiv*. 2016. doi:10.1101/067215.

[43] Nomenclature Committee of the International Union of Biochemistry and Molecular Biology: *Enzyme Nomenclature 1992: Recommendations*. Academic Press. 1992. ISBN 0122271645.

[44] O'Donovan, C.; Martin, M. J.; Gattiker, A.; et al.: High-quality protein knowledge resource: SWISS-PROT and TrEMBL. *Briefings in bioinformatics*. vol. 3, no. 3. 2002: pp. 275–284.

[45] Orengo, C. A.; Todd, A. E.; Thornton, J. M.: From protein structure to function. *Current opinion in structural biology*. vol. 9, no. 3. 1999: pp. 374–382.

[46] Pavlová, M.: *Screening and in vitro construction of environmental biocatalysts [online]*. Disertační práce. Masarykova univerzita, Přírodovědecká fakulta, Brno. 2009 [cit. 2017-05-10].
Retrieved from: http://is.muni.cz/th/12799/prif_d/

[47] Pevzner, P. A.; Tang, H.; Waterman, M. S.: An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*. vol. 98, no. 17. 2001: pp. 9748–9753.

[48] Rabiner, L.; Juang, B.: An introduction to hidden Markov models. *ieee assp magazine*. vol. 3, no. 1. 1986: pp. 4–16.

[49] Rabiner, L. R.: A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. vol. 77, no. 2. 1989: pp. 257–286.

[50] Rice, P.; Longden, I.; Bleasby, A.: EMBOSS: the European molecular biology open software suite [software]. Version 6.6.0.0.
Retrieved from: https://www.ebi.ac.uk/Tools/emboss/

[51] Rosypal, S.: *Úvod do molekulární biologie*. Brno: Prof. RNDr. Stanislav Rosypal, DrSc., Brno. Čtvrté inovované vydání edition. 2006. ISBN 80-902562-5-2.

[52] Schleif, D. R. F.: *Genetics and Molecular Biology*. The Johns Hopkins University Press. 1993. ISBN 0801846749.

[53] Smith, T. F.; Waterman, M. S.: Identification of common molecular subsequences. *Journal of molecular biology*. vol. 147, no. 1. 1981: pp. 195–197.

[54] Tipton, K.; Boyce, S.: History of the enzyme nomenclature system. *Bioinformatics*. vol. 16, no. 1. 2000: page 34. doi:10.1093/bioinformatics/16.1.34.

[55] Van Pee, K.-H.; Unversucht, S.: Biological dehalogenation and halogenation reactions. *Chemosphere*. vol. 52, no. 2. 2003: pp. 299–312.

[56] Xiong, J.: *Essential Bioinformatics*. Cambridge University Press. 2006. ISBN 978-0-521-84098-9.

[57] Zerbino, D.: Velvet Manual [online]. Version 1.1. Acessed on 2017-05-19.
Retrieved from: https://www.ebi.ac.uk/~zerbino/velvet/Manual.pdf

[58] Zerbino, D. R.; Birney, E.: Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*. vol. 18, no. 5. feb 2008: pp. 821–829. doi:10.1101/gr.074492.107.

Appendix A

Contents of the CD

- `data/` – data from experiments, example metagenomic data and enzyme datased used in experimental evaluation of verification methods
- `doc/` – LaTeX source files of this text
- `src/` – source files of all parts of the implemented system
- `src/biodb/` – source files of the Biodb Python 3 library, which contains implementations of enzymatic function verification methods with other supporting modules
- `src/metacentrum/` – wrapper scripts, which enable use of the system on a grid computing infrastructure with PBS Pro scheduler
- `lib/` – third party software required for execution of the system
- `install.sh` – installation script
- `INSTALLATION.txt` – installation instructions and a list of dependencies
- `projekt.pdf` – electronic version of this thesis

Appendix B

The Format of Quality Pre-Processing Configuration File

The pre-processing script for Metacentrum, `dispatcher.sh`, expects configuration file in JSON format with information about quality pre-processing as one of its inputs. This file should contain a single JSON object with following attributes:

Attribute	Type	Meaning
Q_leading	Integer	Threshold for leading base cutting.
Q_trailing	Integer	Threshold for trailing base cutting.
Q_sliding	Integer	Threshold for sliding window cutting.
minlen	Integer	Reads shorter than minlen will be automatically discarded.
adapter	String	Absolute path of a fasta file with technical sequences.

For example, a valid configuration file might have the following form:

```
{
  "Q_leading"  :  20,
  "Q_trailing" :  20,
  "Q_sliding"  :  20,
  "minlen"     :  50,
  "adapter"    :  "/home/user/TruSeq3-PE.fa"
}
```

In this case, the quality pre-processing will use threshold 20 for all types of low-quality region cutting, discard reads shorter than 50 nucleotides and use technical sequences from the file `/home/user/TruSeq3-PE.fa` as patterns for technical sequence removal.

Appendix C

The Format of Annotated Enzyme File

In order to perform enzymatic function verification, scripts `verify.sh` and `verify.py` require a sequence of some known enzyme with an annotation of its catalytic regions. File containing this data must be in JSON format and must contain a single JSON object with following attributes:

Attribute	Type	Meaning
sequence	String	Valid amino acid sequence over the IUPAC alphabet.
features	Array	Array of catalytic region annotations.
features[*].pos	Array	Array of two integers; beginning and ending position of a catalytic region.

Each catalytic region annotation is represented by a JSON object with attribute "pos", which is an array of two integers. First integer denotes the position of the first residue in the catalytic region and the second the position of the last residue in the catalytic region. Following figure shows an example of a valid enzyme file:

```
{
      "sequence": "MEFAAFADRAEAIE...",
      "features": [
        {
          "pos": [
            255,
            255
          ]
        }
      ]
}
```

Appendix D

Syntax of the Classification Method String Descriptor

The verification script `verify.sh` expects user to provide an exact specification of the verification method. This specification has form of a classifier descriptor string, which is passed to the script through its command line parameters. Its basic syntax is

NAME[:key1=value1,key2=value2,...,keyN=valueN]

where the string NAME identifies the classification method and the sequence of key-value pairs corresponds to classification parameters. The global alignment scoring method is denoted by the name "global", local by the name "local" and correlation method by the name "correlate". The classification parameters are different for every method and are described in following sections. In the case of correlation method, there are no classification parameters. The classification window size is not specified as part of the descriptor string. If any of classification parameters is omitted, all methods will use its default value. Following is the example of classifier specification:

global:matrix=blosum62,featurew=2,fixed=True

Global Alignment

Key	Value type	Meaning	Default Value
matrix	String	Name of the scoring matrix to use. If set to "None", simple scoring model will be used. List of available matrices via `verify.sh -l`.	blosum62
opengap	Numeric	Penalty for gap opening in alignment.	−0.5
extendgap	Numeric	Penalty for gap extension in alignment.	−0.1
fixed	Boolean	If True, fixed window method will be used. If False, soft window method will be used.	True
featurew	Integer	Scoring weight of the catalytic region.	2
multf	Boolean	If True, the score of a window will be multiplied by the score of a catalytic region.	False

Local Alignment

Key	Value type	Meaning	Default Value
matrix	String	Name of the scoring matrix to use. If set to "None", simple scoring model will be used. List of available matrices via `verify.sh -l`.	None
opengap	Numeric	Penalty for gap opening in alignment.	−0.5
extendgap	Numeric	Penalty for gap extension in alignment.	−0.1

Appendix E

Example Working Session

The main goal of this appendix is to provide a practical example of working session with the implemented system. The tutorial will show usage of all parts of the system, beginning with the read pre-processing and finishing with enzymatic function verification. All files used in this example are available on the attached CD in the directory `data/`.

After the user has obtained raw metagenomic reads, the first important task is their pre-processing. The pre-pocessing script expects paired-read data stored in two separate files – one for forward and one for reverse reads. Both files have to be in fastq format, their names must share a common prefix and end with suffix "_1.fastq" in the case of forward reads, and "_2.fastq" in the case of reverse reads. For example, reads used in this session are stored in files `example_1.fastq` and `example_2.fastq`.

The pre-processing can be run either on a standard computer, using the script `process.sh`, or on a grid computing infrastructure, using the script `dispatcher.sh`. Apart from reads, the user has to specify quality thresholds for low-quality region removal and a fasta file with standard technical sequences, which were used during the sequencing process.

In the case of the script `process.sh`, thresholds are specified as its commandline arguments, and must be provided in the phred quality format. For illustration, consider the following example:

```
process.sh -f example -c 1 -l 30 -t 30 -s 30 -m 50 -a TruSeq3-PE.fa

Output:
out_1.fasta   - processed forward reads
out_2.fasta   - processed reverse reads
single.fasta  - processed single reads
protein.fasta - translated forms of all sequences
```

The first argument of `process.sh` is the prefix of raw read files (`-f`), following is the number of processes to use (`-c`), then the leading (`-l`), trailing (`-t`) and sliding window (`-s`) quality threshold and the final argument is the file with technical sequences (`-a`). In this case, the pre-processing will be done using the quality threshold 30 (99.9% base correctness probability) and technical sequence patterns will be loaded from the file TrueSeq3-PE.fa. This file with technical sequences is available on the attached CD as part of the Trimmomatic [9] software package located in the `lib/` directory.

The same pre-processing result can be achieved using the script `dispatcher.sh` on a grid computing infrastructure. This is illustrated in the following example:

```
dispatcher.sh example 1 2 qual.json

Output:
Directories named 0 and 1, containing the same files as in the case
of the script process.sh.
```

In contrast to the script `process.sh`, quality thresholds and the file with technical sequences are not specified as commandline parameters, but through a special quality configuration file in JSON format. Details about its syntax are discussed in the appendix B.

Regarding other commandline arguments, the first represents the number of processors to use per node and the second is the number of processing nodes. The resulting directory structure will contain as many subdirecotries as there were processing nodes specified. Each subdirectory will contain the same files like in the case of the script `process.sh`.

After the pre-processing, the sample is ready for searching. The search can be conducted either on a personal computer (script `searchdb.sh`) or on a grid (script `metasearch.sh`). First, let us consider example with the personal computer:

```
searchdb.sh -p protein.fasta -n out_1.fasta -n out_2.fasta \
            -n single.fasta dha_tuberculosis_aa.fa

Output:
out_1_filtered.fasta    - matching forward reads
out_2_filtered.fasta    - matching reverse reads
single_filtered.fasta   - matching single reads
```

In this example, the search was conducted using the search query sequence from the file `dha_tuberculosis_aa.fa` over the protein file `protein.fasta`. Files `out_1.fasta` and `out_2.fasta`, specified through the parameter `-n`, must contain untranslated reads corresponding to protein sequences stored in the file `dha_tuberculosis_aa.fa`. If a multiple sequence alignment is used instead of a single sequence, it must be provided as a profile HMM in a format compatible with HMMER. Furthermore, its use must be indicated by parameter `-v`.

The same can be achieved on a grid using the script `metasearch.sh`:

```
metasearch.sh -d . -c 1 dha_tuberculosis_aa.fa

Output:
out_1_filtered.fasta    - matching forward reads
out_2_filtered.fasta    - matching reverse reads
single_filtered.fasta   - matching single reads
```

In this case, the same search will be conducted using metagenomic database stored in the current directory (parameter `-d`) with one cpu (parameter `-c`). The number of workers is equal to the number of subdirectories (chunks) of the metagenomic database.

After the search is finished, found reads have to be assembled into longer sequences. This can be done using the assembly program Velvet. Following figure illustrates its use with the example data:

```
velveth assem 20 -shortPaired -fasta -separate out_1_filtered.fasta \
        out_2_filtered.fasta -short single_filtered.fasta

velvetg assem/ -exp_cov 10 -cov_cutoff 2 -scaffolding no

Output:
assem/contigs.fa -- resulting assembled sequences
```

For further details about Velvet and its use, I refer reader to its manual [57]. Please note that scaffolding must be disabled, otherwise Velvet might add special nucleotides "N" into the file with resulting contigs. These represent placeholders for any nucleotide and are not part of common scoring matrices. Their presence will result in an error.

Finally, enzymatic function verification can be performed:

```
verify.sh -s dha_tuberculosis_aa.fa -c assem/contigs.fa \
          -j tuberculosis.json

Output:
results.fa - File containing resulting found enzymes.
scores.csv - File containing verification scores of matches.
```

In the example, the verification has used annotated known enzyme located in the file `tuberculosis.json` to verify assembled sequences stored in the file `assem/contigs.fa`. The parameter `-s` refers to the input query sequence, which was used in the search. If the search was conducted using a multiple sequence alignment, one of the aligned sequences or consensual sequence of the alignment must be provided. Classification method can be chosen by providing classifier descriptor string through the parameter `-m`. If no method was chosen, cross-correlation will be used. Syntax of this string is further discussed in the appendix D. The syntax of the annotated known enzyme file is presented in the appendix C.

The resulting file `results.fa` contains only found enzymes, which have the same catalytic function as the input query. Threshold for verification can be either set by the user, via the parameter `-t`, or, like in this case, automatically, by the script `verify.sh`. Please note that the automatic setting works only for classifiers with default classification parameters and otherwise defaults threshold to 0.